Approche d'extension de méthodes fondée sur l'utilisation de composants génériques

Rebecca Deneckere

THESE

Pour obtenir le titre de DOCTEUR DE L'UNIVERSITE PARIS I

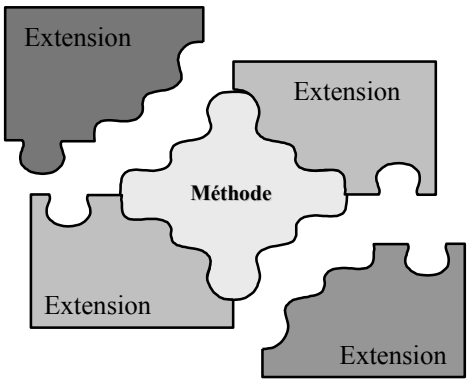
Discipline : INFORMATIQUE

Présentée et soutenue publiquement par

Rébecca DENECKERE

le 4 Janvier 2001

Approche d'extension de méthodes fondée sur l'utilisation de composants génériques



Jury

Colette Rolland	Directeur de thèse
Carine Souveyet	Co-directeur de thèse
Jean-Pierre Giraudin	Rapporteur
Michel Léonard	Rapporteur
Yann Pollet	Membre du jury
Farida Semmak	Membre du jury

A mes parents,
sans eux, rien n'aurait été possible...

Je tiens à remercier en premier lieu les membres du jury :

Michel Léonard, professeur au Centre Universitaire d'Informatique de Genève, et Jean-Pierre Giraudin, professeur à l'université Pierre Mendès France, pour m'avoir fait l'honneur d'être les rapporteurs de mon mémoire.

Colette Rolland, ma directrice de thèse, professeur à l'université Paris1 Panthéon-Sorbonne, pour m'avoir accueillie dans son équipe et pour ses conseils.

Carine Souveyet, ma co-directrice de thèse, maître de conférences à l'université Paris1 Panthéon-Sorbonne, pour sa disponibilité, ses conseils, sa patience et son aide.

Yann Pollet, membre de Matra Systèmes et Informations, et Farida Semmak, maître de conférences à l'université Paris12 Créteil pour avoir accepté de faire partie de mon jury.

Outre les membres du jury, mes remerciements vont à mes collègues de travail de ces quelques années :

Tous les membres de l'équipe du Centre de Recherche en Informatique pour leur aide constante, leur soutien, leur gentillesse et leurs encouragements.

En particulier Jolita, Corinne et Mustapha avec qui j'ai partagé le même espace de bureau durant ces trois dernières années.

Enfin, je n'oublierai pas mes amis et ma famille, en particulier :

Les trois collègues de bureau cités plus haut, qui sont devenus des amis très chers au fil du temps, et sans qui les mauvais moments auraient peut-être été plus forts que les bons.

Mon mari qui a fait beaucoup d'efforts pour me ménager du temps libre et me permettre de terminer ce mémoire.

Mes parents et ma sœur Annick qui ont mis à l'épreuve leurs talents (très grands d'ailleurs) de correcteurs orthographiques et grammaticaux.

Les autres membres de ma famille pour leur soutien au cours de ces années.

Chapitre I : Introduction

1 Domaine de la thèse

La thèse s'inscrit dans le domaine de l'ingénierie des méthodes et plus spécialement dans le domaine de l'ingénierie des méthodes situationnelles.

1.1 Les systèmes d'information

Les systèmes d'information (SI) sont la représentation d'un système du monde réel, une représentation des objets concrets de l'organisation dont ils permettent d'améliorer la gestion. A l'origine ils concernaient certaines fonctions administratives, comme l'administration des paiements ou celle des commandes. Les systèmes d'information scientifiques étaient plutôt utilisés pour appliquer des calculs, par exemple pour gérer la physique nucléaire ou les programmes de recherche spatiale. Cependant, les applications des SI ont beaucoup évolué. Ils peuvent maintenant gérer les interactions entre réseaux (comme les navigateurs Internet ou les outils de vidéoconférences), produire des informations essentielles pour diriger une entreprise ou guider les flots de documents ou bien encore offrir des fonctionnalités combinées comme les outils de multimédia.

Différentes définitions d'un SI peuvent être trouvées dans la littérature ; nous retiendrons la définition suivante [Rolland88]:

"Un système d'information est un artefact qui supporte un réseau de flux d'informations nécessaires pour organiser, mettre en œuvre, gérer et maintenir les activités d'une organisation. C'est un instrument de communication qui sert aux échanges informationnels entre les partenaires de l'organisation et qui accroît leur efficacité."

De nos jours, aucune organisation ne peut fonctionner de façon efficace sans s'appuyer sur un SI collectant, mémorisant, traitant, et communiquant l'information dont elle dispose de manière efficace.

1.2 L'ingénierie des systèmes d'information

Tous les types de modification (correction, extension, amélioration, automatisation, etc.) composent les phases de développement des systèmes d'information permettant de satisfaire les besoins d'une organisation. [Harmsen96] emploie le terme d'*ingénierie des systèmes d'information* pour insister sur le fait que ce processus est contrôlé par des méthodes et des outils.

Le processus de développement des SI est traditionnellement décomposé en plusieurs sous-processus. On distingue le processus d'analyse, le processus de conception et le processus de réalisation. Les produits résultants de ces processus sont respectivement : le schéma conceptuel du système, le schéma interne du système et enfin le système opérationnel (voir Figure 1).

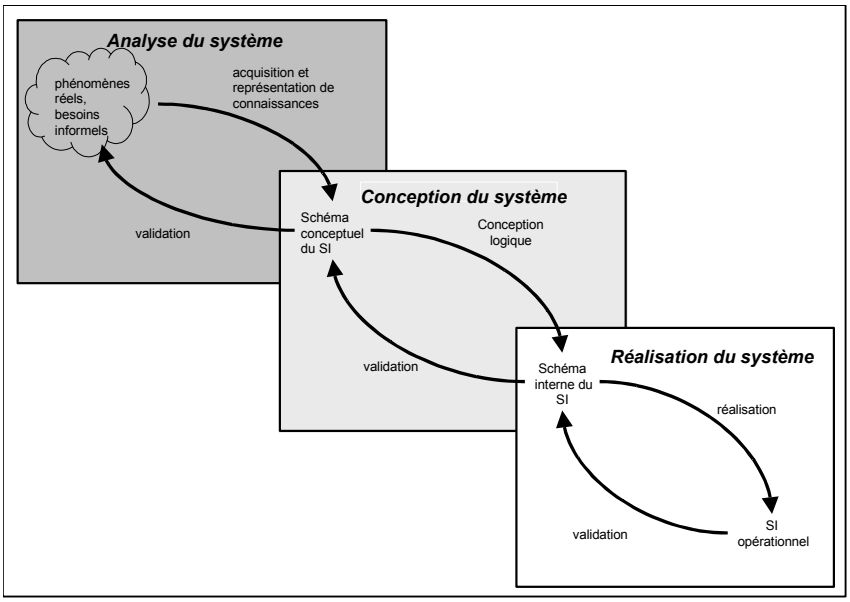


Figure 1: Le processus de développement des SI

L'ingénierie des systèmes d'information a toujours été problématique. De nombreuses études montrent que le développement de ces systèmes est devenu complexe et difficile [Boehm87], [Charrette86], [Turner87], [Flood90]. Les systèmes d'information s'automatisent un peu plus de jour en jour et utilisent donc un nombre croissant de composants logiciels et matériels. Cette automatisation prend donc une importance incrémentale dans la société actuelle. Les types d'applications augmentent rapidement, et touchent virtuellement tous les domaines et les acteurs du SI. En même temps la complexité des SI croît, non seulement du fait de l'augmentation du nombre d'applications mais aussi du fait des exigences qu'on leur impose en terme d'utilisation, d'intégration, d'efficacité, de flexibilité et d'accessibilité. Les projets de développement sont difficiles à gérer, dépassent généralement le temps prévu ainsi que leur budget et produisent des produits difficilement acceptés par les utilisateurs.

Les quatre dernières décennies ont vu fleurir des solutions à tous ces problèmes, la majorité préconisant une standardisation du processus d'ingénierie des SI. Donc, afin de maîtriser les processus d'analyse et de spécifier des systèmes d'information de bonne qualité, les ingénieurs qui en ont la charge ont recours à des méthodes dites *d'analyse*.

1.3 Les méthodes d'analyse

L'analyse et la conception orientée objet ont entraîné un intérêt exceptionnel, à la fois dans le domaine de la recherche et dans l'industrie. Cependant, l'état de l'art concernant les méthodes d'analyse est extrêmement diffus et il devient très difficile d'en suivre les développements.

Le mot « *méthode »* vient du grec « methodos » qui signifie « moyens d'investigation ». Les méthodes d'analyse couvrent tout ou partie du cycle de développement d'un système d'information. Parmi toutes les définitions d'une méthode d'analyse proposées par [Olle91] [Brikkemper90], [Lyytinen89], [Wynekoop93], ([Kronlof93], [Smolander91], [Prakash94], [Seligmann89], [Harmsen94]) nous retiendrons le fait que la plupart d'entre elles convergent vers l'idée suivante :

Une méthode est basée sur un ou plusieurs modèles de produit (systèmes de concepts) et consiste en un ou plusieurs modèles de processus (étapes exécutées dans un ordre donné).

Plus simplement, [Brinkempper96] pose la définition suivante :

Une méthode est une approche permettant de développer un projet, basée sur une certaine manière de penser, constituée de règles, structurée d'une façon systématique dans les activités de développement correspondant aux produits de développement.

Nous pouvons donc dire qu'une méthode donne les concepts pour décrire le produit et les règles de conduite méthodologiques pour produire un produit de qualité avec une efficacité raisonnable. Comme il est dit dans [Olle91], la partie **PRODUIT**[1] est la cible désirée d'un développement de système d'information alors que la partie processus, ou partie **DÉMARCHE**[2], est la route à suivre pour atteindre cette cible.

La partie **PRODUIT** est représentée par différents modèles. Un *modèle de produit* est la notation dans laquelle un **PRODUIT** est décrit. Il est associé à une *méthode* et permet de décrire les produits qui résultent de l'application de la démarche qui lui est associée. Une méthode peut avoir plusieurs *modèles de produit* correspondant à différentes notations. Dans la méthode OMT par exemple, on distingue trois *modèles de produit* : le modèle objet, le modèle dynamique et le modèle fonctionnel. Le modèle de produit est composé d'un ensemble de concepts et de règles pour les manipuler permettant de modéliser la partie **DÉMARCHE** d'une méthode. Le modèle de processus décrit habituellement les étapes à suivre et les actions à entreprendre pour développer un système d'information. En conséquence, une méthode donne les concepts pour décrire le **PRODUIT** et les règles de conduite méthodologiques pour produire un produit de qualité avec une efficacité raisonnable.

La partie **DÉMARCHE** représente le processus à accomplir pour définir le produit. Cette partie est l'ensemble cohérent des activités permettant la construction d'un système d'information. Il représente l'ensemble des étapes et des heuristiques permettant de spécifier les décisions à prendre, comment les prendre et dans quel ordre. Une *démarche,* ou un *processus* est, dans la majorité des cas, « un ensemble d'activités inter-reliées et menées dans le but de définir un produit » [Franckson91]. Un *Modèle de processus* est l'abstraction d'une classe de processus. Il décrit les étapes à suivre et les actions à entreprendre pour développer un système d'information. Dans la communauté des systèmes d'information, un modèle de processus correspond à la démarche méthodologique prescrite par la méthode utilisée.

Durant les vingt dernières années, les recherches relatives à l'analyse des SI se sont focalisées sur la définition de modèles pour représenter les différentes facettes des SI (structurelle, dynamique et fonctionnelle) à différents niveaux de détail. Parallèlement de nouveaux paradigmes et de nouvelles applications ont émergé. Il en a résulté un grand nombre de modèles, chacun ayant ses forces et ses faiblesses. [Cris82] et [Olle88] présentent un large éventail de ces modèles.

[1] Pour plus de clarté dans ce mémoire, le mot « produit » est écrit différemment pour représenter la « partie **PRODUIT** ».
[2] De même, le mot « démarche » est écrit différemment pour représenter la « partie **DÉMARCHE** ».

Parmi les premières méthodes publiées se trouvent SA/SD [DeMarco78] ou encore YSA [Yourdon79]. En général, ces méthodes se concentrent uniquement sur certaines étapes du processus de développement et ont tendance à négliger le processus général du projet. De plus, ces méthodes sont principalement centrées sur le développement des SI administratifs traditionnels.

Au milieu des années 80, des méthodes incluant de plus grandes parties du processus général du projet ont commencé à apparaître, telles que Merise [Tardieu86] et SSADM [Longworth92]. Des méthodes permettant de couvrir d'autres domaines d'application que les traditionnels SI administratifs furent proposées, par exemple le développement de systèmes à base de connaissances avec KADS [Tansley93] [Schreiber93] ou le développement de systèmes temps réel avec [Hatley87] et [Ward85]. Des outils ont été développés afin d'aider les utilisateurs à appliquer les méthodes et permettre leur meilleure acceptation. Il existe différents types d'outils. Les AGL[3] assurent des fonctionnalités d'édition, de stockage et de documentation des produits mais ils n'intègrent qu'exceptionnellement la démarche de la méthode. Des outils CASE[4] [McClure89] ont également été crées pour les méthodes de développement des SI.

Ces dernières années, de nouveaux paradigmes et domaines d'applications ont émergé, tels que l'orientation objet, la gestion des flux de données ou le multimédia.

L'orientation objet, provenant des langages orientés objets tels que SmallTalk et C++, est le paradigme ayant le plus de succès, ceci étant certainement dû à la grande quantité de méthodes orientées objet telles que OMT [Rumbaught91], OODA [Booch91], OOSE [Jacobson93], OOA&D [Martin92], OSA [Shlaer88, 92] ou OOAD [Coad90, 91]. Les méthodes proposées sont d'une certaine façon immatures, à la fois théoriquement et empiriquement. Leurs théories sont pauvrement documentées et un certain nombre d'entre elles sont juste des assemblages de concepts de langages de programmation. Elles combinent des concepts orientés objets aux systèmes d'information traditionnels et des concepts de génie logiciel comme la modélisation E/R à la modélisation des flots de données. Leurs expériences pratiques sont très limitées et la plupart d'entre elles sont sujettes à de fortes évolutions provoquées à la fois par les considérations du marché et l'expérience pratique.

Il existe des analyses comparatives de méthodes [De Champeaux91], [Iivari94], [Goor92] qui utilisent surtout des critères de comparaison dérivés de la tradition OO. Cependant, même si ce type d'évaluation interne montre les différences et les similitudes des méthodes, il ne donne pas beaucoup d'information sur l'application des méthodes OOA&D comme moyen de développement de SI [Wynekoop93]}.

La diversification des applications et leur complexité grandissante, demandent beaucoup aux méthodes d'analyse. Elles doivent accepter la standardisation, gérer la complexité mais doivent être également assez flexibles pour anticiper de nouveaux types d'applications. Certaines ont été proposées, et sont encore proposées, couvrant de nouveaux domaines d'applications. Cependant, définir une méthode complète est un processus de longue haleine et elles sont souvent déjà dépassées lors de leur

[3] Ateliers de Génie Logiciels
[4] Ingénierie des systèmes assistées par ordinateurs

publication. De plus, elles ne peuvent jamais s'appliquer complètement à un projet d'ingénierie des SI car chaque projet est différent des autres et les méthodes sont, par définition, rigides et standardisées. Chaque système d'information est développé pour une situation spécifique. Chaque situation est différente. Il n'y a en fait pas de méthode qui puisse s'adapter à toutes les situations. D'un côté, elle doit accomplir une standardisation, et d'un autre côté elle doit rester flexible pour s'adapter à la situation. La définition d'une méthode compréhensive et de bonne qualité est un long processus devant partir du principe que chaque projet est différent et qu'elle doit s'accorder à la situation.

Une étude de 3 ans de Ernst & Young concernant la pratique des méthodes dans des projets de développement de systèmes d'information, a montré qu'une large partie des 35% des efforts gaspillés dans les projets est due à l'utilisation de méthodes standard de développement car tous les projets sont différents et nécessitent donc des spécificités différentes que ne leur offrent pas ces méthodes standards. Suivant le même raisonnement, Parkisson [Parkisson96] affirme que « *Les méthodologies ont tendance à traiter tous les projets comme s'ils étaient identiques, alors qu'en pratique, chacun est différent. En traitant tous les projets de la même manière, les méthodologies conduisaient les chefs de projet à créer des plans de travail contenant des tâches inutiles, ou l'absence de valeur supplémentaire pour un projet particulier* ».

Pour toutes ces raisons, les méthodes actuelles ne permettent pas aux usagers d'être guidés efficacement dans leur travail, de partager et de réutiliser leurs expertises de manière systématique.

Il est à noter que les méthodes d'analyse traditionnelles (OMT, OOA&D [Martin92], Merise [Tardieu83, Tardieu85], etc.) ne sont pas construites de façon à s'adapter à une situation en cours mais de manière à résoudre les problèmes que l'on trouve dans les applications courantes. Elles sont développées de façon à résoudre les problèmes posés dans certaines situations types. Malheureusement, les applications sont souvent très spécifiques et comportent des points qui ne sont pas pris en compte par ces méthodes. Il faut donc trouver un moyen d'intégrer la prise en compte de ces points délicats dans la méthode d'origine (dans la partie **PRODUIT** comme dans la partie **DÉMARCHE**). C'est à cette problématique qu'est due l'arrivée de l'*ingénierie des méthodes*.

1.4 Ingénierie des méthodes

Pour pallier les limites que nous venons d'évoquer, une nouvelle approche de définition des méthodes a vu le jour : *l'ingénierie des méthodes*. Nous utiliserons la définition suivante de [Kumar92] :

« L'ingénierie des méthodes est *la discipline visant à construire et adapter une méthode de développement de SI et ses outils associés à chacun des projets spécifiques auxquels elle est appliquée. »*

Font partie de cette nouvelle discipline :

Les techniques de méta modélisation pour la construction de méthodes ;

L'interopérabilité des outils avec, par exemple, la définition d'environnements CAME[5] guidant les usagers et permettant de capitaliser les connaissances heuristiques liées à l'application des méthodes [Harmsen94] ;

Les méthodes situationnelles pour l'intégration de méthodes [Kronlof93] ;

Les comparaisons de méthodes [De Champeaux91], [Goor92], [Iivari94], [Song92], [Hong93] et d'outils [Martiin93].

L'ingénierie des méthodes s'est créée en réponse à l'impression de plus en plus importante que les méthodes ne sont pas adaptées [Lyytinen87] aux besoins des utilisateurs, les concepteurs d'applications. Cette discipline vise à construire et à adapter méthodes, techniques et outils dans le développement des systèmes d'information. En particulier, il est nécessaire de modifier les méthodes d'une application à l'autre [Hidding94]. L'ingénierie des méthodes [Welke92] [Harmsen94] représente l'effort fait pour améliorer l'utilisation des méthodes d'analyse en créant un cadre d'adaptation où les méthodes sont créées pour correspondre à des situations organisationnelles spécifiques. Ceci a été fait à deux niveaux. A un niveau global, elle gère la détermination des facteurs de contingence du projet [VanSlooten96][Euro96] aidant à sélectionner la bonne méthode à utiliser. A un niveau plus fin, elle gère la construction du processus pour la situation en cours.

Une méthode doit accomplir une standardisation, mais elle doit cependant rester flexible pour prendre en compte les spécificités des situations rencontrées. Harmsen [Harmsen94] utilise le terme de flexibilité *contrôlée* pour définir cette exigence.

Cette flexibilité est obtenue, dans la majorité des approches d'ingénierie des méthodes, en définissant une phase préliminaire dans le projet consistant à caractériser la situation du projet de manière globale puis, à l'aide de cette caractérisation, de composer une méthode adaptée à cette situation.

L'Ingénierie des méthodes a vu se développer un nouveau groupe de méthodes appelées *situationnelles* qui permettent de construire et d'adapter une méthode d'analyse aux projets spécifiques auxquels elle est appliquée.

Une méthode situationnelle [Kumar92] est une construction de plusieurs méthodes selon des situations particulières de développement de projets. Chaque développement de système débute alors par une phase de définition de méthode où la méthode de développement est construite.

Ces constatations mettent en évidence le besoin de mieux comprendre la notion de méthode, de savoir les décrire, d'être capable de les représenter, de les modéliser, mais aussi de les construire et de les utiliser dans différents contextes de projet.

[5] CAME : Computer Aided Method Engineering

2 Problématique de la thèse

Ce travail se situe dans le domaine de l'ingénierie des méthodes situationnelles. Notre champ d'investigation porte sur la modification de méthodes pour une adaptation à la situation réelle de l'application en cours. En effet, le domaine de l'ingénierie des méthodes situationnelles a émergé en réponse à la sensation croissante que les méthodes n'étaient pas bien adaptées. Différentes techniques d'utilisation de méthodes existent et il est possible de les classer selon certains critères. Les différentes études ([Harmsen94]) faites à ce sujet montrent qu'il est possible de modifier ces méthodes selon le projet en cours et les besoins de l'ingénieur de méthodes pour les adapter à des situations données en tenant compte de leurs spécificités.

Il est donc possible d'utiliser ce type de technique pour permettre une modification du **PRODUIT** et de la **DÉMARCHE** d'une méthode spécifique selon la situation en cours et les besoins de l'ingénieur de méthodes.

La modification, comme la construction d'une méthode, est un travail complexe et difficile car peu de méthodes sont réellement adaptées à l'application désirée et il est nécessaire à l'ingénieur de méthodes de définir clairement les besoins de l'application qui ne peuvent être résolus par la méthode en cours. Cela nécessite une bonne connaissance à la fois de l'application et de la méthode. Une fois les besoins définis, il faut ensuite appliquer plusieurs opérations de modification sur la méthode dans le but de l'adapter à la situation. Il est cependant possible que plusieurs méthodes puissent être modifiées de la même manière, par la même suite d'opérations, car elles ont une partie de **PRODUIT** ou de **DÉMARCHE** qui leur sont identiques. Pour une meilleure optimisation du travail, il peut donc être intéressant de conserver cette connaissance pour pouvoir la réutiliser ultérieurement pour d'autres méthodes.

Il est par conséquent nécessaire de proposer un mécanisme de stockage de la connaissance.

Il est à noter qu'il faut également gérer la cohérence de la méthode après modification. En effet, toute altération des concepts ou de la démarche de construction d'une méthode peut engendrer des problèmes de cohérence importants. Chaque modification doit donc permettre de gérer ce problème. Cependant, il est également possible que certaines modifications ne puissent se suivre de façon aléatoire car possédant entre elles une certaine relation de précédence garantissant la conservation de cette cohérence. Exécuter une modification avant l'autre pourrait induire des erreurs de modélisation catastrophiques pour la suite du développement.

Il est donc nécessaire de pouvoir organiser les modifications selon leur ordre de précédence.

Après analyse de plusieurs cas possibles (plusieurs méthodes classiques, plusieurs situations particulières, plusieurs aspects différents à intégrer, etc.), nous avons été amenés à penser que la connaissance nécessaire aux modifications était construite de façon générique, selon certains critères spécifiques. Il est donc possible de définir des règles pour un critère donné et pour un type de modification donné qui seront applicables quels que soient la méthode à modifier ou le besoin à combler.

Il est par conséquent possible de proposer des techniques génériques de construction de ces opérations de modification quels que soient la situation et les besoins.

3 Aperçu de la solution proposée

L'objectif de ce travail est de proposer une approche d'amélioration des méthodes existantes selon la situation en cours par une technique d'extension grâce à des patrons spécifiques.

Ces patrons doivent satisfaire les conditions suivantes :

- Ils doivent être génériques pour pouvoir être applicables à toute méthode d'analyse orientée objet ;
- Ils doivent utiliser les principes de la réutilisation pour leur conception et leur utilisation (les mêmes patrons peuvent être réutilisés dans le processus d'extension de plusieurs méthodes différentes) ;
- Ils doivent s'appliquer dans une situation particulière pour satisfaire une intention particulière (ils sont orientés objectif).

Pour satisfaire nos objectifs nous proposons :

- Une approche de représentation de la connaissance concernant les extensions de méthodes sous forme de patrons décrits par différents langages de description et de manipulation ;
- Une technique d'organisation de ces patrons, pour faciliter le guidage lors de leur réutilisation, par la spécification d'un modèle de processus décisionnel exécutable appelé carte d'extension ;
- Une approche de conception des patrons d'extension à l'aide de méta-patrons génériques.

Le processus d'extension correspondant à la solution proposée est illustré Figure 2.

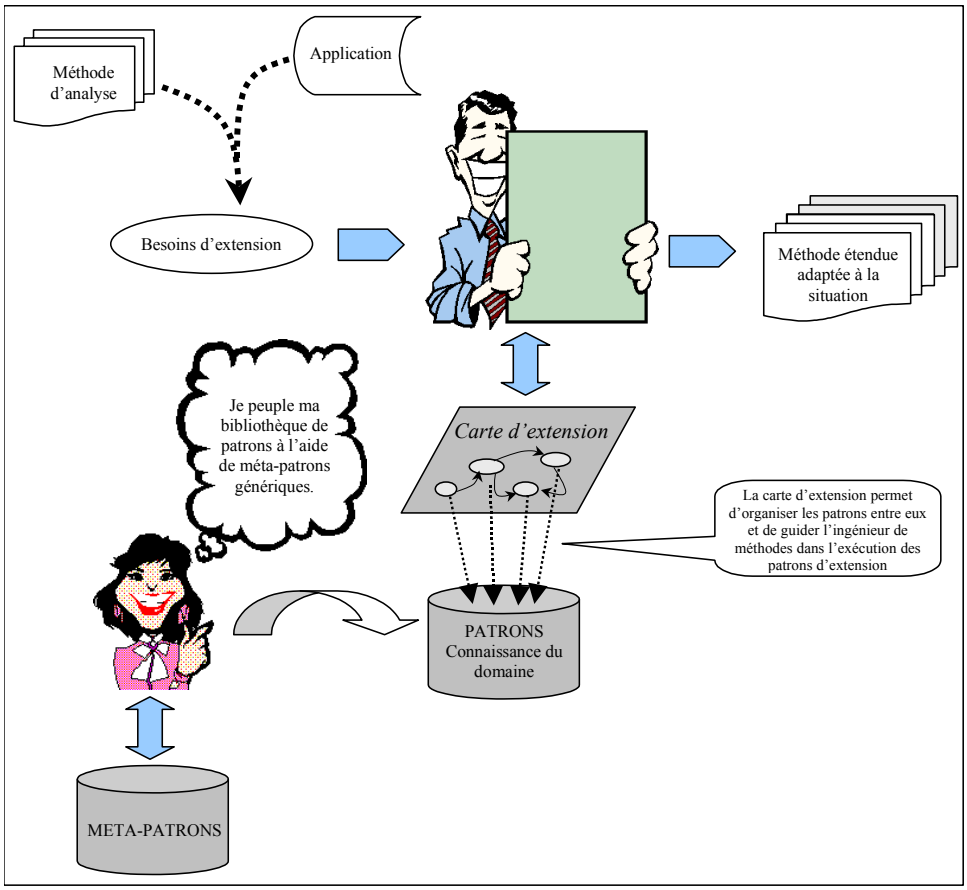


Figure 2 : Processus d'extension

Cette solution propose donc de stocker la connaissance relative aux extensions dans une bibliothèque spécifique appelée *PATRONS*, peuplée grâce à des patrons de construction génériques eux-mêmes stockés dans la bibliothèque appelée *META-PATRONS*. Les patrons d'extension sont ensuite organisés entre eux par le biais des *cartes d'extension* que l'ingénieur de méthodes utilise pour produire une méthode étendue adaptée à ses besoins d'extension.

4 Plan du mémoire

Chapitre II : Présentation du cadre de référence ainsi que d'un état de l'art sur l'ingénierie des méthodes par rapport à ce cadre de référence.

Chapitre III : Introduction de la solution proposée.

Chapitre IV : Description de la structure formelle des patrons d'extension.

Chapitre V : Définition de l'organisation des patrons par le biais des cartes d'extension.

Chapitre VI : Illustration de la construction des patrons d'extensions grâce à la définition de patrons génériques appelés « méta-patrons ».

Chapitre VII : Illustration de la solution par l'extension de la méthode O* aux applications temporelles.

Chapitre VIII : Présentation de la conclusion.

Chapitre II : Ingénierie des méthodes, état de l'art

Ce travail se place donc dans le cadre de l'ingénierie des méthodes. Nous allons présenter dans cette section le cadre de référence utilisé dans ce mémoire, ce qui nous permettra de pratiquer une évaluation de plusieurs approches par rapport à ce cadre de référence.

Cadre de référence de l'ingénierie des méthodes

Plusieurs aspects peuvent être différenciés dans le domaine de l'ingénierie des méthodes. Ces aspects peuvent être définis comme autant de critères différents permettant de décrire une méthode. Nous considérerons successivement les critères concernant les différentes approches de construction, la flexibilité de ces approches, les techniques de construction possibles, les modèles de représentation de la connaissance, les différentes manières de construire celles-ci et de les organiser.

4.1 , pproches de construction de méthodes

D'une façon similaire à l'utilisation du spectre des méthodes situationnelles, les approches de construction de méthodes peuvent être décrites suivant leur flexibilité. En effet, les méthodes peuvent être construites en utilisant soit une approche rigide, soit une approche par contingence.

4.1.1 , pproche statique

Les approches *statiques* consistent à prescrire entièrement et statiquement la méthode. Ces approches produisent des méthodes rigides, complètement prédéfinies, difficilement adaptables à la spécificité de la situation rencontrée.

4.1.2 , pproche par contingence

Les approches *par contingence* sont celles utilisées par les méthodes situationnelles. Elles consistent à définir des facteurs de contingence pouvant se définir sur une application en cours de développement. La méthode utilisée est construite en fonction de la valeur de ces facteurs, à partir d'une base de modules réutilisables pouvant répondre à ceux-ci.

Une approche de construction peut donc être classée selon sa nature définie comme suit :

```
Nature : ENUM {statique, par contingence}
```

4.2 Flexibilité des approches de construction de méthodes

Le spectre des méthodes situationnelles de [Harmsen94] organise les approches d'ingénierie de méthodes situationnelles selon leur degré de flexibilité à s'adapter aux besoins situationnels. Ces approches sont ensuite placées sur une échelle de flexibilité (allant d'une flexibilité « Faible » à « Elevée »). Sur la partie basse de cette échelle se trouvent les méthodes dites « rigides » alors que la partie haute contient les approches de construction de méthodes modulaires [Harmsen94]. La Figure 3

illustre le fait que l'on peut regrouper les approches d'ingénierie des méthodes dans trois grands types différents : les approches rigides, semi-rigides et situationnelles.

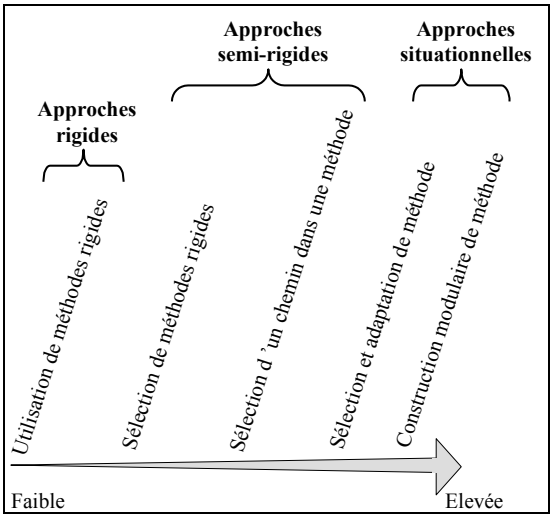


Figure 3 : Spectre des approches d'ingénierie des méthodes basé sur [Harmsen94]

4.2.1 , pproche rigide

Les approches *rigides* utilisent des méthodes complètement pré-définies laissant peu de possibilités pour les adapter à la situation rencontrée. Elles adoptent certaines standardisations basées sur un type de projet et ne fournissent pas d'aide concrète pour adapter ces méthodes à d'autres types de projets. Des méthodes comme SSADM [Longworth92] ou Merise [Tardieu83] appartiennent à cette catégorie.

4.2.2 , pproche semi-rigide

Entre les approches rigides et les méthodes situationnelles existe un certain nombre d'approches que l'on peut appeler *semi-rigides* [Benjamen99]. Ce sont des approches qui permettent la sélection dans un panel de méthodes ou chemins rigides (sélection basée sur la situation du projet). Ensuite, la méthode sera utilisée telle quelle sans adaptation. Il est en général peu vraisemblable que la méthode sélectionnée réponde à toutes les exigences du projet. Cette typologie comprend également les approches proposant plusieurs chemins à suivre selon le type du projet en cours. Dans ce type d'approches se trouvent par exemple Toolkit [Nenyon87] et Multiview [Wood-Hgarper85] qui se résument en l'inclusion de plusieurs méthodes, chacune concernant un aspect spécifique du système. Selon la situation, l'une de ces sous-méthodes sera employée dans le projet. Cependant, cette approche ne fournit pas de suggestions concrètes pour les facteurs qui déterminent la sélection de ces outils. De plus, la compatibilité des sous-méthodes incorporées est problématique [Zaal92].

4.2.3 , pproche situationnelle

Par contre, les *approches situationnelles* utilisent et/ou modifient des méthodes pour les adapter à une situation donnée en tenant compte de ses spécificités.

Les approches par sélection et adaptation d'une méthode permettent, pour chaque nouveau projet, de sélectionner des méthodes parmi plusieurs et de les accorder aux besoins du projet (modification, extension, suppression, etc.). Les approches par la construction de méthodes modulaires permettent de créer des méthodes facilement décomposables en modules, ce qui facilite leur réutilisation pour la construction d'autres méthodes.

Une approche de construction de méthode situationnelle peut donc être classée selon sa flexibilité définie comme suit :

```
Flexibilité : ENUM {rigide, semi-rigide, situationnelle}
```

4.3 Techniques de construction de méthodes

Il y a différentes façons de construire une méthode. Les constructions de méthodes utilisent la notion de *méta modèle* et se placent dans plusieurs catégories principales bien définies : l'instanciation, l'assemblage, l'intégration et l'extension.

4.3.1 Construction par instanciation

Comme toutes les applications sont spécifiques, de nouvelles méthodes ont souvent besoin d'être définies. Une manière de procéder est d'identifier les caractéristiques communes et génériques des méthodes et de les représenter ensuite par un système de concepts appelé *méta modèle*. Une telle représentation a l'avantage de permettre la création de toutes les méthodes partageant ces mêmes propriétés (cf. Figure 4).

Une telle approche se construit en identifiant les systèmes de concepts génériques puis en définissant les techniques d'instanciation à partir de ce méta modèle. L'exploitation de celui-ci aide à définir plusieurs méthodes différentes. Cela incite les constructeurs à trouver des solutions génériques aux problèmes traités puisque les méthodes ainsi dérivées hériteront des caractéristiques de leur solution.

Dans cette approche par instanciation, le problème crucial n'est plus celui des méthodes mais celui du méta modèle. Cela signifie que la responsabilité de fournir une bonne méthode n'incombe plus au concepteur de méthode mais au concepteur du méta modèle.

La technique d'instanciation a été utilisée dans [Rolland93], [Rolland94], [Rolland96]. Cette technique a également été utilisée pour construire les bases de connaissance des Ateliers d'Ingénierie des Méthodes [Kelly96], [Harmsen95], [Merbeth91], [Si-said96], [Souveyet91].

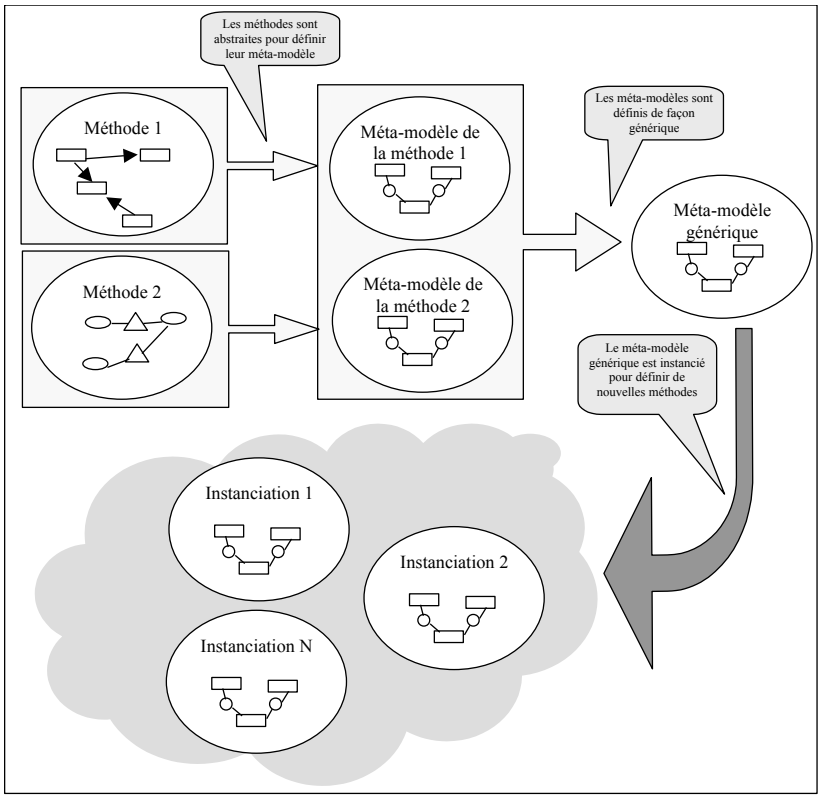


Figure 4: Construction par Instanciation

4.3.2 Construction par assemblage

Les approches par assemblage se concentrent généralement sur le groupement de fragments de méthodes appartenant à des méthodes se complétant les unes les autres [Brinkkemper98] [Song95]. Elles permettent donc de gérer l'assemblage de fragments disjoints de méthodes choisis par rapport au projet spécifique étudié et que l'on assemble pour former une méthode unique.

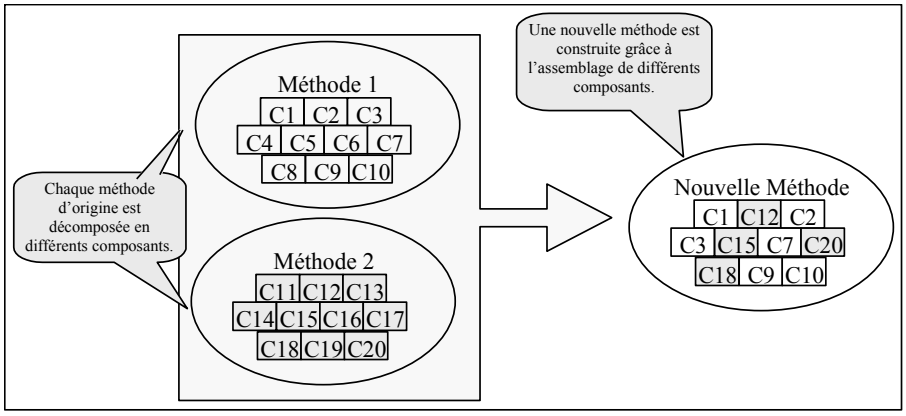


Figure 5: Construction par Assemblage

Beaucoup de méthodes peuvent être considérées comme le résultat de l'application d'une méthode d'assemblage [Brinkkemper98]. Par exemple, la méthode OMT a été construite à partir des fragments existants suivants : diagramme de classes d'objets (diagramme d'entités-relations étendu), diagramme d'états-transitions, graphes de séquences de messages et diagramme de flots de données, tous

provenant d'autres méthodes sources. Cet exemple montre que les méthodes d'assemblage produisent une nouvelle méthode puissante qui peut modéliser des systèmes compliqués à partir de plusieurs points de vue : vue objet, vue comportementale, et vue fonctionnelle. En conséquence, les méthodes d'assemblage sont une technique importante pour construire à la fois des méthodes situationnelles et des méthodes puissantes ayant des points de vue multiples.

Le projet IDE[6] est un autre exemple représentatif puisqu'il utilise une base de fragments de méthodes contenant des morceaux venant des méthodes SSADMv4 [Longworth92] pour la méthode de base, OMT pour l'analyse et la conception orientée objet et Navigator [Ermst&Young91] pour les étapes et les produits concernant la sélection et l'implémentation des paquetages. Ces morceaux sont utilisés dans le but de construire une méthode correspondant au projet en cours.

Pour que la technique d'assemblage soit réussie, il est nécessaire que les modèles de processus soient modulaires. Si la technique d'assemblage est combinée avec la technique d'instanciation, alors le méta modèle doit lui même être modulaire.

4.3.3 Construction par intégration

Des approches [Ralyté99] gèrent le problème de l'intégration de méthodes qui se chevauchent partiellement. Par exemple, deux méthodes peuvent contenir le même concept mais ayant un sens différent. Elles se rapprochent donc du problème de l'intégration de schémas dans le domaine des bases de données [Bouzeghoub90]. Pour assembler ces deux méthodes différentes, il est nécessaire d'avoir un ensemble de techniques spécifiques permettant de réaliser une intégration préservant l'intégrité de la méthode.

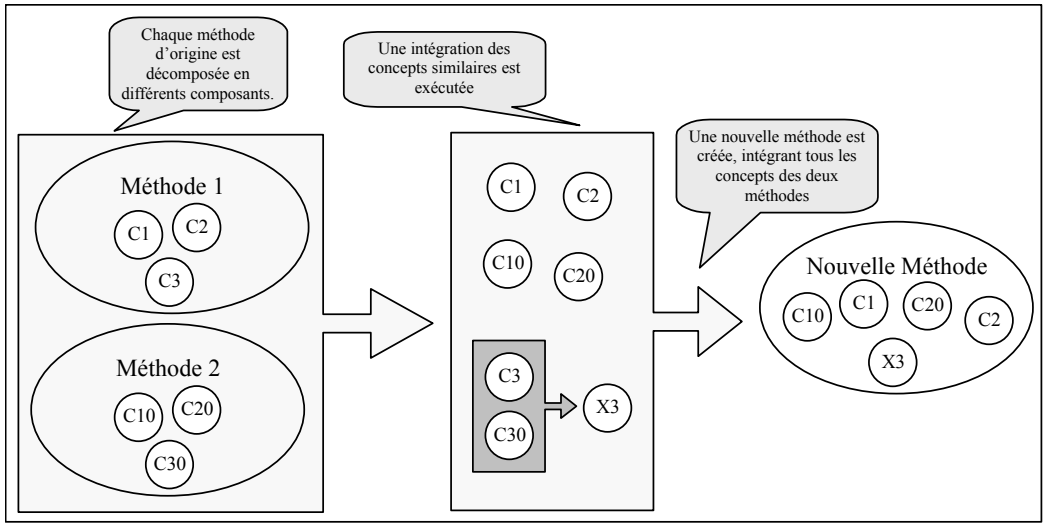


Figure 6: Construction par Intégration

[6] IDE : ICPI development environment (ICPI : Industrial Process Control Inforamtion systems (compagnie mondiale de développment de SI pour les applications en temps réel)

Par exemple, le projet Esprit CREWS[7] utilise cette approche pour intégrer les techniques de scénarios dans différentes méthodes (qu'elles contiennent ou non certains concepts inhérents à ces techniques).

4.3.4 Construction par extension

Une approche par extension permettra d'étendre une méthode rigide pour en faire une méthode adaptée aux besoins de l'ingénieur de méthodes.

Les méthodes construites par extension se rapportent au dernier type d'approche selon la classification de Harmsen, celle de la construction modulaire. Les approches par extension ne se basent que sur une seule méthode, que l'on appellera la *méthode d'origine*. Celle-ci devient le corps d'une *méthode étendue* répondant à des besoins spécifiques grâce à de nouveaux concepts, ainsi que l'illustre la figure suivante.

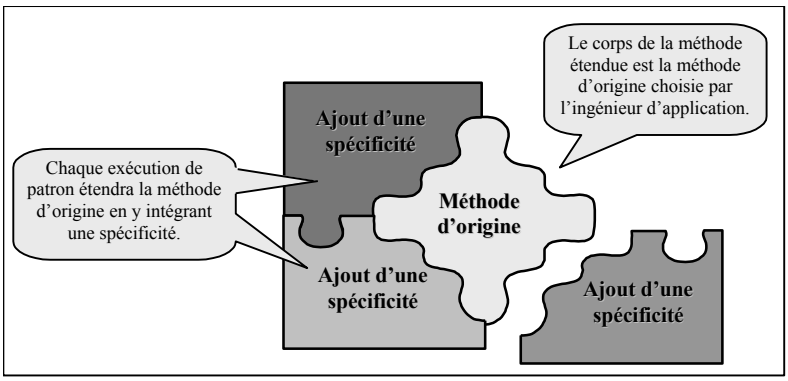


Figure 7: Principe d'extension de méthode

Ce type de construction diffère donc des approches vues précédemment par le fait que la méthode prise en compte avant les modifications est unique. En effet, les trois autres approches (par instanciation, par assemblage et par intégration) n'ont la possibilité d'être exécutées que par rapport à plusieurs méthodes d'origine.

De plus, dans le cadre d'une extension, les spécificités à insérer sont déjà présentes dans une base de connaissance spécifique, ce qui permet leur ajout dans la méthode, alors que les approches précédentes n'utilisent pas d'autres composants que ceux des méthodes d'origine concernées.

Un type de construction peut donc être classé selon sa technique définie comme suit :

```
Technique: SET OF {instanciation, assemblage, intégration, extension}
```

4.4 Modèles de représentation de la connaissance

Les méthodes situationnelles utilisent différentes techniques pour représenter la connaissance : les fragments, les composants et les patrons.

[7] CREWS est un acronyme pour Cooperative Requirements Engineering With Scénario. Projet Esprit 21 903.

4.4.1 Fragments de méthode

Les méthodes de construction de méthodes situationnelles basées sur les fragments consistent à encourager une analyse globale des projets rencontrés en se basant sur des *critères de contingence.* Les projets et les situations sont caractérisés à l'aide de facteurs qui sont associés aux méthodes ou aux composants. [VanSlooten96] en est un exemple. Cette approche a été expérimentée dans neuf projets non-standards dans le département de développement du domaine bancaire. La méthode adaptée à la situation est construite à l'aide de ces fragments. La construction est soutenue par des règles d'assemblage de composants et des contraintes devant être satisfaites par la méthode ainsi construite. Les règles et contraintes font partie de la connaissance emmagasinée dans la base de méthodes pour soutenir le processus de construction de la méthode adaptée.

4.4.2 Composants de méthode

Les méthodes de construction de méthode situationnelle basées sur les composants visent à associer à ces composants réutilisables des *descripteurs* afin de faciliter la recherche et l'extraction des composants selon les besoins des utilisateurs. [Rolland96b] utilise la notion de descripteur [DeAntonellis91] comme un moyen pour décrire des composants de méthode, (par exemple les fragments de processus). De la même manière que les composants de méthode, les descripteurs sont organisés d'une manière contextuelle : chacun d'eux définit la situation dans laquelle le composant peut être employé et décrit l'intention de son usage. La partie intention référence les caractéristiques des projets de développement dans lesquelles le composant peut être utilisé comme une partie de la méthode du projet. Ce type de représentation a été expérimenté sur les méthodes de construction des systèmes d'information [Plihon95] dans le projet NATURE et les bases de connaissances accessibles via Internet dans les approches basées sur les scénarios dans le projet CREWS [Rolland98].

4.4.3 Patrons de méthode

Le concept de patron a été très largement utilisé dans la communauté du développement logiciel durant les deux dernières décennies, en particulier dans le domaine des approches orientées objet. Ce concept a été plus récemment introduit dans la communauté de l'ingénierie des méthodes. Ce récent intérêt concernant les patrons trouve son origine dans [Coad92] puis est passé à la programmation logicielle [Beck97][Bushmann96], la conception système et logicielle [Coplien95][Gamma94] [Vlissides96], la modélisation des données [Hay96] et plus récemment dans l'analyse des systèmes [Fowler97]. Tous ces efforts conduisent à réutiliser les meilleurs atouts d'une méthode dans une autre.

Une grande partie du travail contemporain sur les patrons a été inspirée par le travail de C. Alexander qui a écrit une étude [Alexander79] sur l'utilisation de ceux-ci dans le domaine de l'architecture des bâtiments. Cet architecte a développé l'idée d'un *langage de patrons* permettant à des personnes non spécialistes de construire leurs propres maisons. Celui-ci est formé d'un ensemble de patrons dont chacun décrit comment résoudre un problème particulier de la construction, problèmes pouvant être larges ou très spécialisés. Ce langage ne nécessite pas de connaissance spécifiques et se concentre sur des problèmes communs ou moins communs lors de la construction d'une maison. Cependant, ce livre montre l'importance des patrons de telle façon que ceux-ci vont bien au-delà du domaine de l'architecture. Alexander présente dans son livre les arguments majeurs pour la découverte des patrons

et leur utilisation. Dans [Alexander77] un patron est décrit comme *"un problème qui se produit souvent dans notre environnement et qui décrit alors le cœur de la solution à ce problème, d'une telle façon que l'on peut utiliser cette solution un million de fois, sans toutefois faire la même chose deux fois. »* Ici, l'importance est mise sur le fait qu'un patron décrit un problème récurrent avec sa solution associée. Le patron fournit une solution et cette solution devient réutilisable pour toute situation relevant de ce problème.

C'est une dizaine d'années après les travaux d'Alexander que la notion de patron a été introduite dans le domaine de l'informatique avec [Beck87]. Depuis, les recherches sur les patrons dans le domaine de l'informatique sont de plus en plus nombreuses [Gamma94] [Pree94] [Coad96] [Front97] [Fowler97] [Rieu99].

L'utilité d'un concept tient premièrement dans le fait qu'il permette aux expériences passées d'être disponibles facilement. Ceci rend le concept de patron idéal dans les projets qui impliquent la conception et ses décisions. En développant des patrons, les utilisateurs condensent une partie de leurs connaissances sur le domaine du problème et permettent sa disponibilité pour les autres utilisateurs. En fait, un patron peut permettre de stocker les meilleures connaissances d'un domaine, connaissances que l'on a déjà prouvé efficaces, et de les communiquer. Lorsque d'autres utilisateurs réutiliseront ce patron, ils bénéficieront de l'expérience des précédents concepteurs.

Dans la communauté de l'ingénierie des méthodes, les patrons génériques proposent un moyen de construire des méthodes selon des situations spécifiques. De tels patrons encapsulent la connaissance sur les démarches pouvant être réutilisées et appliquées dans différents cas. Ils permettent de guider les ingénieurs de méthodes dans la construction de méthodes.

Les types de représentation de la connaissance peuvent donc être classés selon leur modèle défini comme suit :

```
Modèle: ENUM {fragment, composant, patron, inexistant}
```

4.5 Construction de la connaissance

La construction traditionnelle de la connaissance, lorsqu'elle existe, est l'expression de l'expérience des ingénieurs d'application. Tant que cette expérience n'est pas formalisée et ne constitue donc pas une connaissance de base disponible pour les différentes applications, on peut dire que cette connaissance est le résultat d'une technique de construction ad hoc. Ceci a deux conséquences majeures : la méconnaissance de la manière dont a été effectuée la construction et la dépendance par rapport au domaine d'expertise. Si cette connaissance doit être indépendante du domaine d'expertise et rapide à construire, il est alors nécessaire de sortir du cadre des techniques de construction basées sur l'expérience pour utiliser des techniques plus formalisées.

Les techniques de construction peuvent donc être classées selon leur construction définie comme suit :

```
Construction: ENUM {formalisée, ad hoc, inexistante)
```

4.6 Organisation de la connaissance

La connaissance utilisée lors de la construction des méthodes peut être stockée de manière à être réutilisée ultérieurement. Souvent, elle est insérée dans des environnements de type bibliothèque.

Ceux-ci fournissent les fonctions de base pour la gestion d'un référentiel de composants. Comme ces bibliothèques peuvent contenir un grand nombre de composants, elles offrent en général des techniques de recherche comme les techniques d'indexation et l'utilisation de mots-clefs. Cependant, de tels outils ne supportent pas une recherche de composants à partir d'un problème de développement donné car les techniques de recherche mises en place nécessitent d'avoir une bonne idée du composant souhaité. En fait, vis à vis du processus d'utilisation de la connaissance pour une application, ces bibliothèques apparaissent comme de simples fournisseurs de composants dont l'accès est explicitement demandé par l'ingénieur.

D'autres approches, en plus du formalisme d'extraction des composants, possède un processus d'organisation de ces composants spécifique au domaine d'application. En effet, il est parfois nécessaire de gérer la cohérence de la connaissance que l'on utilise par rapport aux méthodes modifiées. En effet, toute altération de méthode existante peut générer des problèmes de cohérence importants. Les processus d'organisation permettent donc de gérer ce problème de façon plus formelle.

Les techniques de construction peuvent donc être classées selon leur organisation définie comme suit :

```
Organisation: SETOF {inexistant, bibliothèque, processus d'organisation}
```

5 Evaluation des approches d'ingénierie des méthodes par rapport au cadre de référence

Cette section présente une évaluation de six approches de construction de méthodes utilisant le cadre de référence défini dans la section précédente. Les approches retenues sont les suivantes.

- L'approche d'intégration par méta-modélisation [Hollegersberg99] ;
- L'approche par le langage de modélisation M-Telos [Nissen99] ;
- L'approche de caractérisation de projet décrite dans [Van Slooten96] ;
- L'approche basée sur les bases de méthodes de [Saeki93] ;
- L'approche des fragments de méthodes développée dans [Harmsen94] ;
- L'approche d'assemblage de composants de méthodes de [Ralyté01].

Cette partie décrit brièvement ces approches puis les situe par rapport au cadre de référence décrit dans la première section de ce chapitre et qui peut se résumer comme suit.

```
Nature : ENUM {statique, par contingence, dynamique}
Flexibilité : ENUM {rigide, semi-rigide, modulaire}
Technique : SET OF {instanciation, assemblage, intégration, extension}
Modèle : ENUM {fragment, composant, patron}
Construction : ENUM {formalisée, ad hoc, inexistante}
Organisation : SETOF {inexistant, bibliothèque, processus d'organisation}
```

Figure 8: Résumé du cadre de référence

5.1 , pproche d'intégration par méta-modélisation

L'approche définie par [Van Hillegersberg99] propose de construire une définition des concepts OO par la méta-modélisation des méthodes existantes pour les intégrer ensuite dans un méta-modèle unique qui définit ces concepts et leurs relations. Plusieurs applications de ce méta-modèle ont été proposées, dont le développement et la sélection de méthodes, mais l'objectif principal de cette approche est de définir et d'intégrer des concepts, tant pour le domaine des langages orientés objet (OOP) que pour celui de l'analyse et de la conception orientées objet (OOAD). D'un point de vue théorique, cette approche fournit une base semi-formelle pour l'intégration dans ces domaines alors que d'un point de vue pratique, elle peut être utilisée pour la sélection et l'évaluation de méthodes, l'ingénierie des méthodes et la correspondance entre une méthode OOAD avec un langage OOP.

Le méta-modèle décrit dans cette approche permet donc de décrire une grande variété des concepts OOP et OOAD par une définition semi-formelle et une validation des utilisateurs dans le but de produire une définition testée et utile des concepts OO. Ce méta-modèle est appelé *Object Modeling Framework* (OMF). L'application de ce méta-modèle est illustrée à la figure suivante.

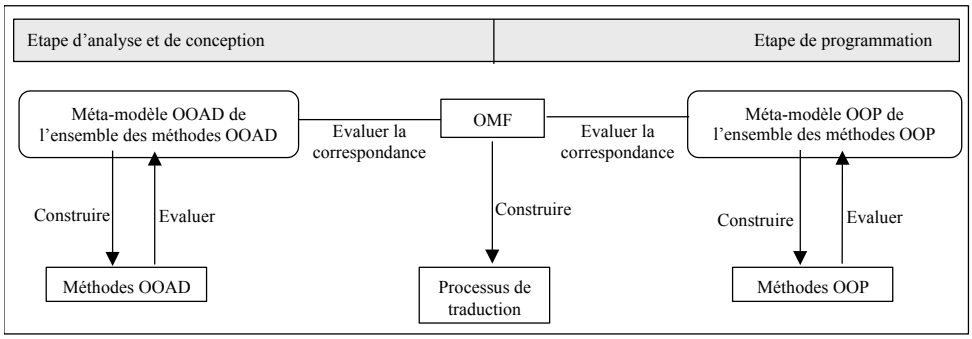


Figure 9: Applications des méta-modèles OMF pour l'évaluation et les processus d'ingénierie des méthodes

Le processus de définition d'un OMF est une séquence composée des cinq étapes suivantes.

Tout d'abord, il y a sélection des méthodes formant les bases de l'OMF et regroupement de celles-ci en deux ensembles différents, l'un portant sur les méthodes OOAD et l'autre sur les méthodes OOP [Jackson83].

Ensuite, chacune des méthodes sélectionnées est méta-modélisée par l'utilisation de la méthode de méta-modélisation GOPRR (décrite par [Welke92] puis étendue par [Tolvanen93]).

La troisième étape représente la validation des méta-modèles obtenus à l'étape précédente, tout d'abord de façon interne par la vérification de la complétude et de la cohérence, puis de façon externe par un panel de trois experts ainsi que par la mise à disponibilité des documents sur Internet pour obtenir des commentaires de personnes extérieures.

Il y a ensuite l'intégration de ces méta-modèles en deux méta-modèles spécifiques : un méta-modèle des méthodes OOAD et un méta-modèle des méthodes OOP. GOPRR a été étendu ici pour faciliter l'intégration par le stockage de détails sur le processus d'intégration (par exemple avec des références aux correspondances entre les différents langages). Des règles de similarités ont été définies pour déterminer les différences entre les différents méta-modèles et pouvoir les résoudre (conflits de noms ou de structure).

Enfin, le processus se termine par la construction de l'OMF avec l'intégration de ces deux méta-modèles et la validation de celui-ci par un examen d'un panel d'experts externes et une révision par des utilisateurs.

Cette approche permet donc de construire une définition d'un méta-modèle permettant de rapprocher les définitions des méthodes OOAD et OOP en construisant un passage entre ces deux types de méthodes. Elle a une nature ***statique*** puisque la définition du méta-modèle permet de construire des méthodes complètement prédéfinies non adaptables à de nouvelles situations. Elle est ***semi-rigide*** puisqu'elle se place dans le cadre des approches utilisant une sélection de méthodes rigides par rapport au projet en cours. Elle utilise la méta-modélisation pour permettre la construction des méthodes par ***instanciation***. Cependant, elle ne se concentre que sur l'aspect de la construction et ne se préoccupe donc pas de celui de la réutilisation de la connaissance. En conséquence, elle n'est pas représentée par un modèle particulier et il n'y a donc pas de construction ni d'organisation de celle-ci. Cette approche se place donc de la façon suivante dans le cadre de référence défini dans la première section.

```
Nature : statique
Flexibilité : semi-rigide
Technique : instanciation, intégration
Modèle : inexistant
Construction : inexistante
Organisation : inexistant
```

5.2 , pproche par le langage de modélisation M-Telos

L'approche proposée par [Nissen99] est basée sur l'utilisation du langage de modélisation M-Telos. Celui-ci intègre les avantages d'adaptabilité et d'analyse du langage Telos (langage de méta-modélisation basés sur la logique) avec un concept de module permettant l'utilisation des mécanismes de structuration des architectures logicielles. Cette approche est basée sur la structure de référence IRDS (*ISO Information Resource Dictionary System Framework* [ISO90]) dont la caractéristique la plus frappante est la possibilité de gérer tous les niveaux d'un cadre de référence logique.

Le processus de cette approche est illustré à la figure suivante.

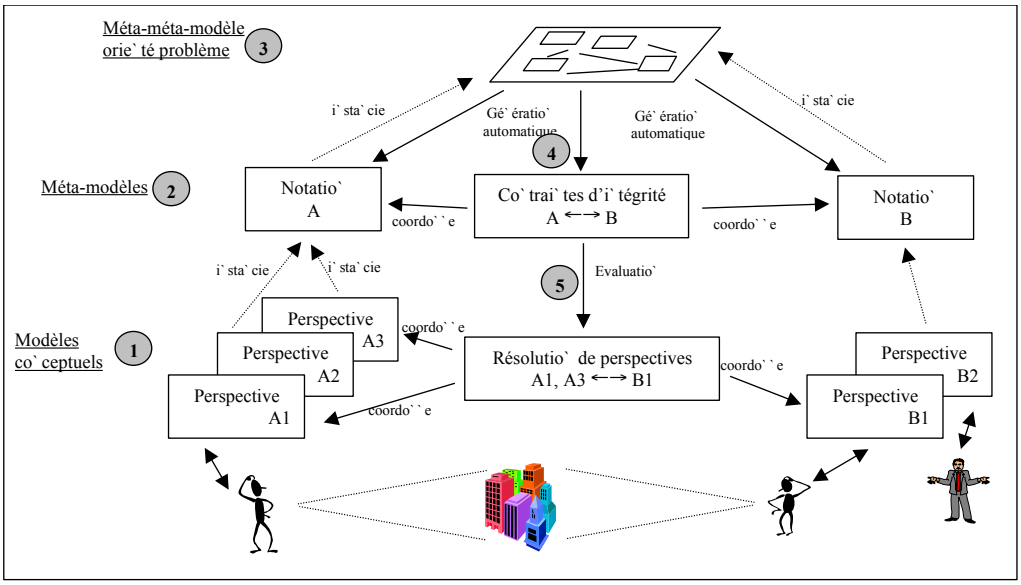


Figure 10: Processus de l'approche basée sur le langage de modélisation M-Telos

Les cinq caractéristiques de cette approche sont les suivantes.

Il y a séparation des perspectives multiples. En effet, chaque modèle conceptuel représente une vue individuelle de la réalité et le mécanisme de séparation offre des contextes de modélisation indépendants. M-Telos utilise le concept de module comme une structure de stockage pour des parties de modèles, ce qui permet la séparation des perspectives et des points de vue conflictuels.

M-Telos utilise des notations personnalisables. En employant les capacités de méta-modélisation extensibles de Telos, on autorise la définition et la modification des méta-modèles dans les notations choisies.

Il y a spécification adaptable des objectifs d'analyse. Un méta-méta-modèle partagé inter-relie les notations de modélisation employées. Elle spécifie la structure du domaine et les objectifs spécifiques de l'analyse. Comme Telos permet une hiérarchie de classification illimitée, même le méta-méta-modèle est modifiable (les objectifs d'analyse peuvent aisément être adaptés aux objectifs du projet spécifique).

L'analyse des perspectives est orientée objectif. Les objectifs d'analyse spécifiés dans le méta-méta-modèle sont définis indépendamment d'une notation spécifique et sont formulés exclusivement dans les termes du domaine. Ces objectifs sont ensuite automatiquement transformés en contraintes d'intégrité sur les méta-modèles de notation.

Pour finir, il y a tolérance des conflits entre perspectives. En effet, pour éviter d'interrompre le processus de modélisation en présence de perspectives inconsistantes, une analyse croisée est effectuée dans des modules de résolution séparés. Un mécanisme basé sur des techniques de bases de données déductives a été développé pour cela.

Cette approche propose donc une solution d'aide outillée pour la modélisation orientée objectifs dans le cadre de perspectives multiples (travail d'équipe). Elle est basée sur un langage de représentation

modulaire de la connaissance, une analyse orientée objectifs des différentes perspectives, une spécification adaptable des objectifs d'analyse et la gestion continue des inconsistances. Cette approche peut être classifiée comme ayant une nature ***statique*** car elle ne produit que des méthodes complètement prédéfinies par l'***instanciation*** du méta-modèle. Elle est caractérisée comme ***semi-rigide*** car elle utilise plusieurs méthodes pour développer les différentes perspectives mais ne les modifie en rien. Elle ne se concentre toutefois que sur l'aspect construction de méthodes et non sur l'aspect réutilisation de la connaissance utilisée. Elle peut se placer dans notre cadre de référence de la manière suivante.

```
Nature : statique
Flexibilité : semi-rigide
Technique : instanciation
Modèle : inexistant
Construction : inexistante
Organisation : inexistant
```

5.3 , pproche de caractérisation de projet

L'approche proposée dans [Van Slooten96] permet de construire une méthode en sélectionnant des méthodes semi-rigides décomposées en modules. Il y a deux sortes de composants à deux niveaux différents de granularité, appelés respectivement *fragment de carte d'itinéraires* et *fragments de méthode*. Un *fragment de carte d'itinéraires* est « une partie d'une carte complète d'itinéraires d'une méthode de développement de système » à laquelle peut être reliés des *fragments de méthode* étant « des parties cohérentes d'une méthode pour le développement de système ou la gestion de projet ». Une *carte d'itinéraires* permet d'établir une approche complète pour un projet ou une méthode adaptée à la situation.

De plus, Van Slooten propose un modèle de *contingence* basé sur le travail de [Van Der Hoef95] avec 17 facteurs pouvant prendre des valeurs comprises entre 'Faible' et 'Elevée'. Ces facteurs sont : l'engagement de gestion (pour le projet du SI), l'importance (du projet), l'impact (du projet), la résistance et conflit (à quel point les participants ont des intérêts différents ou s'opposant), les contraintes de temps, la pénurie de ressources humaines, la pénurie de moyens, la formalité (des procédures du projet), la connaissance et l'expérience(de l'équipe de projet), les compétences, la taille, les rapports (entre le SI en cours de développement et le Si existant), la dépendance (du projet avec des facteurs externes), la clarté (des buts du projet, des objectifs etc.), la stabilité (des buts du projet) et le niveau d'innovation. De même, des contraintes permettent d'affecter l'approche du projet et peuvent être considérées comme des facteurs de contingence spécifiques.

Van Slooten a donc définit une approche ***par contingence*** puisqu'elle aide à la caractérisation du projet à l'aide de facteurs spécifiques permettant de sélectionner et d'assembler les composants de méthode appropriés à un projet (elle a été testée dans neuf projets non-standards dans le département de développement de systèmes d'une organisation bancaire). Elle est donc ***situationnelle*** puisqu'elle permet de créer une méthode modulaire adaptée au projet en cours par l'***assemblage*** de différents ***composants***. Cependant, le processus d'assemblage n'est pas décrit de manière détaillée et cette approche n'explicite pas la façon de créer les composants réutilisables. Ceux-ci ne sont pas non plus

stockés pour une réutilisation ultérieure. Cette approche peut donc être caractérisée de la façon suivante dans le cadre de référence définit dans la première section de ce chapitre.

```
Nature : par contingence
Flexibilité : situationnelle
Technique : assemblage
Modèle : composant
Construction : inexistante
Organisation : inexistant
```

5.4 , pproche basée sur les bases de méthodes

L'approche détaillée dans [Saeki93] utilise des méta-modèles pouvant être instanciés pour permettre de construire des méthodes. Elle permet de concevoir des méthodes rigides qui ne peuvent pas s'adapter à différentes situations et il est nécessaire de créer une nouvelle méthode pour chaque nouvelle situation rencontrée. La méthode de base sélectionnée par l'ingénieur de méthodes est stockée dans une base de méthodes permettant le choix de celle-ci, selon la situation, dans un panel de plusieurs méthodes.

Cette approche définit un méta-modèle de **PRODUIT** et un méta-modèle de **DÉMARCHE**. Ces deux méta-modèles sont décrits à la Figure 11. Comme nous pouvons le voir sur cette figure, le méta-modèle de **PRODUIT** possède les six concepts types communs à la majorité des méthodes (Etat, Evénement, Processus, Donnée, Objet, Association) ainsi que les six concepts types de relations (entrée, sortie, suit, a-un , est-un, définit). Le méta-modèle de **DÉMARCHE** consiste en un concept de Procédure et de quatre concepts de relations (entrée, sortie, a, précède).

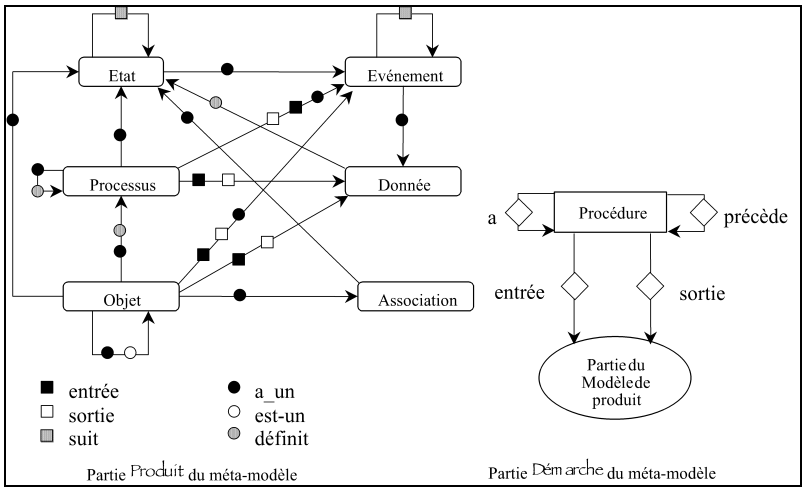


Figure 11: Méta-modèles de produit et de processus.

Les composants de méthode définis grâce à ces méta-modèle sont également de deux types différents : *produit* et *processus.*

Cette approche est ***statique*** car les méthodes produites sont complètement prédéfinies et ne sont pas adaptables à une autre situation. Elle est ***semi-rigide*** puisque les méthodes d'origines sont

sélectionnées dans un panel de méthodes rigides. L'utilisation de la méta-modélisation permet de générer des méthodes par ***instanciation***. Les modules utilisés dans cette approche sont des ***composants*** de méthodes qui ne sont cependant pas caractérisés pour une réutilisation ultérieure (pas de stockage ni d'organisation des composants). Le tableau suivant résume les caractéristiques de cette approche par rapport à notre cadre de référence.

```
Nature : statique
Flexibilité : semi-rigide
Technique : Instantiation
Modèle : composant
Construction : inexistante
Organisation : inexistant
```

5.5 , pproche des fragments de méthodes

L'objectif de cette approche [Harmsen94] est d'adapter des méthodes en sélectionnant et en assemblant ce qui est appelé des *fragments de méthodes* (produit, activité ou outil faisant parti d'une méthode existante). Les méthodes sont décomposées en fragments qui sont ensuite stockés dans une *base de méthodes* permettant leurs réutilisations.

Les fragments doivent être capable de décrire chaque aspect des méthodes et celles-ci sont composées de deux parties : la partie **PRODUIT** et la partie **DÉMARCHE**. En conséquence, deux types de fragments doivent être définis : les *fragments de produit* pour capturer l'aspect **PRODUIT** de la connaissance de méthodes (documents, modèles, diagrammes, concepts, etc.), et les *fragments de démarche* pour capturer l'aspect **DÉMARCHE** lié à cette connaissance (étapes, activités, tâches, etc. devant être exécutées). Les fragments peuvent être considérés au niveau conceptuel comme au niveau physique (description d'outils de stockage et représentation des processus). Ces deux types sont respectivement appelés les *fragments conceptuels* et les *fragments techniques*.

Les fragments ont entre eux des relations d'instanciation, de précédence (uniquement au niveau des fragments de démarche), d'informatisation (un fragment conceptuel est informatisé par un fragment technique), d'entrée-sortie (quels sont les fragments nécessaires à, ou produits par, d'autres fragments). Ces différents types de relations permettent de vérifier la consistance de la méthode. De plus, les fragments ont d'autres propriétés comme la représentation du produit, l'objectif et la signification d'une description, etc. Toutes ces caractéristiques ont besoin d'être prises en compte lors du processus d'assemblage permettant d'intégrer ces fragments les uns avec les autres.

Les opérations de manipulation nécessaires à la construction des méthodes sont de trois types : administration, sélection et assemblage. Le processus d'assemblage de cette approche est décrit à la Figure 12 comme un diagramme de flux de données. Il débute avec la caractérisation de l'environnement du projet qui inclut l'organisation de développement de systèmes existants, l'organisation du client, l'organisation du fournisseur, le domaine de l'application, l'information et la politique d'informatisation, etc. Les facteurs contextuels et de contingence dérivés de l'environnement du projet servent à sélectionner les composants de méthode appropriés au projet pour permettre la construction de la méthode situationnelle adaptée. Ils sont déterminés pendant la caractérisation du

projet à l'aide d'entrevues, de réunions, de questionnaires, etc. Les facteurs importants sont utilisés pour sélectionner des composants appropriés dans la base de méthodes.

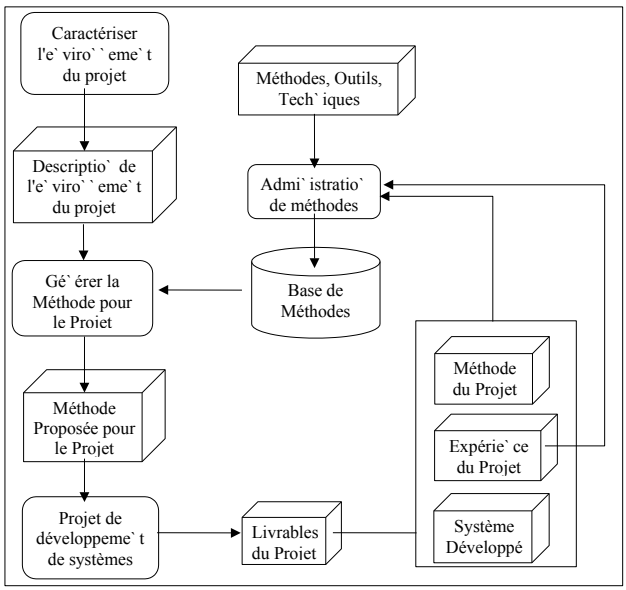


Figure 12: Processus de construction des méthodes situationnelles

La méthode adaptée à la situation est construite à l'aide de ces fragments. La construction est soutenue par des règles d'assemblage de composants et des contraintes devant être satisfaites par la méthode ainsi construite. Les règles et contraintes font partie de la connaissance emmagasinée dans la base des méthodes pour soutenir le processus de construction de la méthode adaptée.

Une fois la méthode définie, le projet de système d'information peut commencer en utilisant la méthode adaptée construite dans la phase précédente et les livrables du projet sont ensuite produits. Les évaluations pendant et après l'exécution du projet peuvent engendrer de nouvelles connaissances pour le développement de méthode adaptée qui sont alors capturées dans la base de méthode. Enfin, de nouveaux composants de méthode peuvent être ajoutés dans la base pour des utilisations ultérieures.

L'approche des fragments de méthodes est une approche ***situationnelle*** car elle se place dans le cadre des approches permettant la construction de méthodes modulaires. Elle utilise un certain nombre de facteurs de contingence permettant la caractérisation de la situation en cours, c'est donc une approche ***par contingence***. Cette technique permet de générer des méthodes modulaires par ***assemblage*** de ***fragments*** de méthodes. Ces modules sont stockés dans une ***bibliothèque*** spécifique appelée *base de méthodes* et crées de manière ***ad hoc***. Cette approche peut donc s'évaluer dans notre cadre de référence de la manière suivante :

```
Nature : par contingence
Flexibilité : situationnelle
Technique : assemblage
Modèle : fragment
Construction : ad hoc
Organisation : bibliothèque (« base de méthodes »)
```

5.6 , pproche d'assemblage de composants de méthodes

L'approche décrite dans [Ralyté01] propose une représentation et une construction des méthodes par assemblage de composants de méthodes. Cette approche est illustrée à la Figure 13.

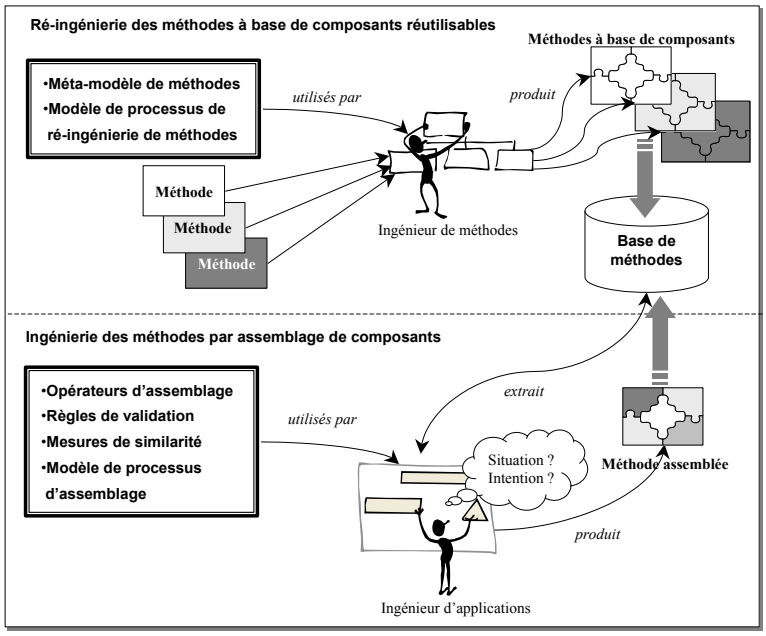


Figure 13: Processus d'ingénierie et de ré-ingénierie proposés par [Ralyté01]

Comme l'illustre la figure précédente, cette approche propose un méta-modèle permettant de créer des méthodes modulaires à l'aide de composants ayant une description situationnelle et intentionnelle. Chacun de ces composants est une partie du processus proposé par la méthode et réutilisable en dehors de celle-ci. Ce modèle de processus modulaire est obtenu par l'utilisation d'un méta-modèle de processus multi-démarche appelé *carte* [Rolland99] [Benjamen99].

Cette approche propose également un processus de ré-ingénierie des méthodes permettant de transformer une méthode en un ensemble de composants réutilisables appliquant ce méta-modèle. Ce processus est basé sur la décomposition de la carte de processus de la méthode en sous-processus appelés *directives*. Celles-ci sont associées aux parties de produit utilisées et forment alors un composant de méthode. Ceux-ci sont décrits à l'aide d'une signature et d'un descripteur permettant leurs identification et leur accès dans la base de méthodes. Cette base permet de les stocker de manière indépendante.

L'approche propose ensuite un modèle de processus d'assemblage de ces composants basé sur l'application d'opérateurs d'assemblage (tant au niveau du Produit que de la Démarche). Un ensemble de règles de qualité sont également proposées dans le but de valider la cohérence et la complétude de la méthode obtenue. De plus, pour une efficacité optimale, certains opérateurs sont utilisés pour mesurer la similarité des éléments des différents composants. L'application de ce processus d'assemblage permet d'obtenir une nouvelle méthode instance du méta-modèle proposé. Celle-ci est donc par conséquent elle aussi une méthode modulaire pouvant être stockée dans la base de méthodes.

L'approche proposée par [Ralyté01] a une nature ***par contingence*** puisqu'elle caractérise la situation de l'application en cours. Elle est ***situationnelle*** puisqu'elle permet de construire des méthodes modulaires par ***instanciation*** et ***assemblage*** de ***composants*** de méthodes. Ces composants sont stockés dans une ***bibliothèque*** appelée *Base de méthodes*. Ils sont construits à l'aide d'un processus ***formalisé*** appelé *Carte*. Cette approche peut donc se placer comme suit dans notre cadre de référence :

```
Nature : par contingence
Flexibilité : situationnelle
Technique : instanciation, assemblage
Modèle : composant
Construction : formalisée (« carte »)
Organisation : bibliothèque (« base de méthodes »)
```

5.7 Résumé de l'évaluation

L'évaluation de ces approches peut être découpée en deux points spécifiques : l'aspect construction de méthodes et l'aspect réutilisation de la connaissance nécessaire à ces constructions.

5.7.1 Construction de méthode

L'aspect construction de méthodes comprend trois facettes particulières : la nature de l'approche (statique ou par contingence), sa flexibilité (rigide, semi-rigide ou situationnelle) et la technique de construction (instanciation, assemblage, intégration, extension). Les six approches peuvent se placer dans le cadre de référence définit dans la première partie de ce chapitre de la manière suivante.

	Méta-modélisation	**M-Telos**	**Caractérisation de Projet**	**Base de méthodes**	**Fragments de méthodes**	**Assemblage de composants**
Nature	Statique	Statique	Par contingence	Statique	Par contingence	Par contingence
Flexibilité	Semi-rigide	Semi-rigide	Situationnelle	Semi-rigide	Situationnelle	Situationnelle
Technique	Instanciation, Intégration	Instanciation	Assemblage	Instanciation	Assemblage	Instanciation, Assemblage

Figure 14: Récapitulatif des approches sous l'aspect Construction de méthodes

Les approches Méta-modélisation, M-Telos et Base de méthodes ont une nature *statique* car elle consistent à prescrire entièrement et statiquement les méthodes construites. Au contraire des autres approches dont la nature est définie comme des approches *par contingence*. En effet, des facteurs de contingences sont définis dans chacune de ces approches pour caractériser le projet en cours de développement.

Les approches sont divisibles en deux groupes distincts : les semi-rigides et les situationnelles. Les approches Méta-modélisation, M-Telos et Base de méthodes sont *semi-rigides* car elles donnent la possibilité de sélectionner des méthodes dans un certain panel et ne les modifient pas. De leur côté, les approches Caractérisation de projet, Fragments de méthodes et Assemblage de composants sont des approches *situationnelles* car elle permettent de construire des méthodes modulaires à base de composants réutilisables.

Les techniques de construction varient selon les approches. Un grand nombre d'entre elles utilisent l'instanciation car l'utilisation des méta-modèles est très répandue. L'approche Méta-modélisation construit un méta-modèle qu'elle *instancie* ensuite pour permettre l'*intégration* des méthodes OOAD et OOP. Les approches M-Telos et Base de méthodes utilisent également un méta-modèle utilisé par *instanciation* pour construire des méthodes spécifiques au projet en cours. L'approche Assemblage de composants construit un méta-modèle de méthodes permettant une *instanciation* des composants et un *assemblage* de ceux-ci pour former une méthode adaptée. Les approches Caractérisation de Projet et Fragments de méthodes ne construisent pas de méta-modèle mais définissent des modules encapsulant la connaissance qu'ils *assemblent* ensuite pour former les méthodes.

5.7.2 Réutilisation de la connaissance

Le tableau suivant récapitule les aspects, concernant la réutilisation, de chaque approche en suivant le cadre de référence. Trois autres facettes caractérisent cet aspect : le modèle de représentation de la connaissance (fragments, composants, patron ou inexistant), la construction de ces modules (formalisée, ad hoc ou inexistante) et leur organisation (bibliothèque, processus d'organisation ou inexistante).

	Méta-modélisation	M-Telos	Caractérisation de Projet	Base de méthodes	Fragments de méthodes	Assemblage de composants
Modèle	Inexistant	Inexistant	Composant	Composant	Fragment	Composant
Construction	Inexistante	Inexistante	Inexistante	Inexistante	Ad hoc	Formalisée
Organisation	Inexistant	Inexistant	Inexistant	Inexistant	Bibliothèque	Bibliothèque

Figure 15: Récapitulatif des approches sous l'aspect Réutilisation de la connaissance

Nous pouvons remarquer que certaines approches ne se concentrent que sur l'aspect de la construction de méthodes et pas sur l'aspect de la réutilisation. C'est le cas des approches Méta-modélisation et M-Telos qui utilisent un méta-modèle instancié pour permettre la construction de la méthode adaptée au projet en cours mais ne conserve pas cette connaissance pour des utilisations ultérieures.

De même, les approches Caractérisation de projet et Base de méthodes identifient la connaissance et la formalisent dans des *composants* spécifiques mais ne la conserve pas pour la réutiliser. Elles ne possèdent donc pas de technique d'organisation de ceux-ci. Elles n'ont pas non plus de processus de construction de ces modules.

Contrairement aux précédentes, les approches Fragments de méthodes et Assemblage de composants s'intéressent à l'aspect réutilisation des modules (*fragment* et *composant* respectivement) encapsulant la connaissance. La construction des fragments est faite de manière ad hoc dans la première approche, c'est à dire que ce processus n'est pas formalisé et que les modules sont construits de manière aléatoire suivant l'expérience des développeurs. Dans la seconde approche, le processus de construction est formalisé à l'aide de « cartes ». Cette technique permet de guider l'ingénieur de méthodes lors de l'élaboration de ses composants. Dans les deux cas, les modules réutilisables sont stockés dans une bibliothèque spécifique.

Chapitre III : Solution proposée – Extension de méthodes à base de patrons

Extension de méthode

La solution proposée dans ce mémoire propose une technique de construction dite *d'extension* pour améliorer une méthode rigide. Le dictionnaire Petit Robert [Robert86] nous donne du terme d'extension les définitions présentées ci dessous.

> **Extension**
> Nom féminin apparu en 1361 et basé sur le nom latin " Extensio ", de " extendre " signifiant " étendre ".
> Didactique : action d'étendre, de s'étendre ; son résultat.
> Action de donner à quelque chose une plus grande dimension ; fait de s'étendre. Résultat de cette action.
> Action de donner à quelque chose une portée plus générale, la possibilité d'englober un plus grand nombre de choses.

Figure 16: Définitions d'extension

Nous en retiendrons la définition suivante : Une extension permet d'englober un plus grand nombre de choses dans l'ensemble de départ. Le *processus d'extension de méthode* va donc pouvoir donner les moyens à l'ingénieur de méthodes de représenter un plus grand nombre de phénomènes réels qu'il ne le pouvait, auparavant, avec la méthode qu'il appliquait.

D'autres techniques permettent d'améliorer des méthodes, i.e. les méthodes par assemblage ou encore par intégration (cf. chapitre II). Ces approches permettent généralement de construire une nouvelle méthode modulaire contenant tous les points intéressants de différentes méthodes. De façon un peu différente, l'approche développée dans ce mémoire ne se base que sur une seule méthode, que l'on appellera la *méthode d'origine,* qui devient le corps d'une *méthode étendue* répondant à des besoins spécifiques grâce à de nouveaux concepts. L'adaptation d'une méthode à des besoins particuliers est un domaine qui n'a pas encore été étudié de manière approfondie dans la littérature existante.

Une méthode ayant été définie comme un ensemble à deux parties : la partie ***PRODUIT*** et la partie ***DÉMARCHE***, chaque extension de méthode devra donc prendre en compte l'extension de ces deux parties différentes, ainsi que l'indique la Figure 17.

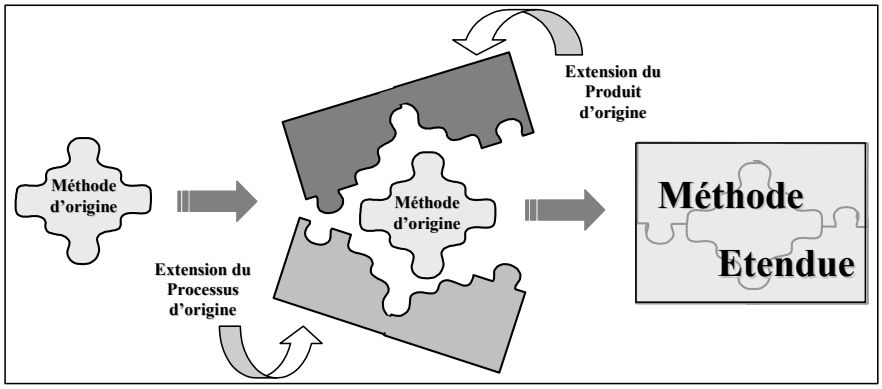


Figure 17: Principe d'extension des parties PRODUIT et PROCESSUS

Cette approche utilise les techniques de construction par ***instanciation*** et par ***extension*** pour permettre de répondre aux exigences du projet en cours (approche ***par contingence***). Elle sélectionne une méthode particulière qu'elle adapte à la situation du projet, elle peut donc être caractérisée comme une approche ***situationnelle***.

Cette solution peut donc se placer dans le cadre de référence des approches de la façon suivante :

```
Nature : par contingence
Flexibilité : situationnelle
Technique : instanciation, extension
```

6 Processus d'extension de méthode

Pour appliquer ce processus, il faut tout d'abord une *méthode d'analyse* classique (dite « rigide » selon la classification de [Harmsen94]), comme OMT, O* [Brunet93] ou encore OOA&D. Dans le but de restreindre notre champ d'investigation, nous nous sommes limités, dans notre expérimentation, aux méthodes orientées objets. Ceci nous a permis de travailler avec des méthodes possédant un grand nombre de concepts communs et où les différences de situation avant extension étaient plus aisées à définir.

Cette méthode peut ne pas contenir certains concepts nécessaires à un bon développement du projet de *l'application* en cours. En effet, même si cette méthode est celle qui est la plus adaptée à ce type d'applications, elle peut très bien ne pas répondre à toutes ses exigences. Prenons l'exemple d'une entreprise dont le département du personnel rythme ses activités sur un calendrier basé sur les jours ouvrables alors que les autres départements utilisent le référentiel classique du calendrier Grégorien. Ce type d'application entraîne des traitements très particuliers, comme le calcul des jours de vacances qui se fera plus facilement avec un référentiel ne contenant que les jours ouvrés que dans un référentiel Grégorien contenant tous les jours de la semaine (où il faudra, par exemple, soustraire les fins de semaines et les jours fériés de toutes les semaines de vacances). Dans ce cas, l'application fonctionnera mieux si l'on y intègre la gestion de plusieurs calendriers spécifiques que l'on ne peut effectuer avec une méthode classique comme OMT. L'ingénieur de méthodes devra donc y intégrer certains concepts temporels, utiles pour le projet en cours mais n'y existant pas à l'origine.

Plusieurs modèles de représentation sont donc possibles pour réaliser ces intégrations de concepts ou de démarches selon les besoins de l'application. Nous avons choisi d'utiliser la technique des ***patrons*** car c'est celle qui nous paraît la mieux adaptée à notre besoin. Ces intégrations sont donc regroupées dans une bibliothèque spécifique sous l'appellation de *Patrons* définis pour guider l'ingénieur de méthodes lors de l'extension de la méthode d'origine.

Cette solution peut donc se placer dans le cadre des approches existantes pour le modèle de représentation comme suit :

```
Modèle : patron
```

En conséquence, l'approche propose une *Bibliothèque de Patrons* permettant d'intégrer certains concepts selon les besoins ressentis par le développeur de l'application. Une méthode étant composée

de deux parties distinctes, c'est à dire la partie **PRODUIT** et la partie **DÉMARCHE**, l'approche propose deux types de patrons différents pour étendre chacune de ces deux parties séparément. La Figure 18 suivante illustre ce processus d'extension de méthode.

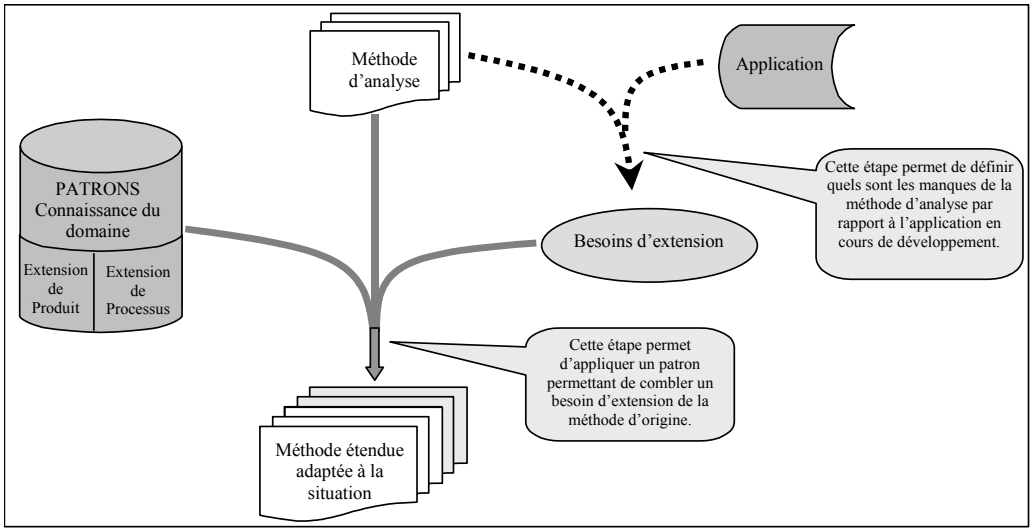


Figure 18 : Processus d'extension

L'extension se fait donc en deux étapes :

Définition des besoins : L'ingénieur doit définir quels sont les *besoins* qu'il aimerait satisfaire mais qui ne peuvent l'être avec la méthode de base pour cause de concepts inexistants ou mal définis.

Extension de la méthode : Une fois le besoin et la situation définis, l'ingénieur de méthodes doit rechercher le patron correspondant, dans la bibliothèque de patrons, puis l'instancier pour étendre la méthode d'origine et obtenir une *méthode adaptée* à ses exigences.

Les patrons d'extension ainsi définis sont stockés dans une bibliothèque spécifique et sont organisés entre eux par le biais de la technique des cartes d'extension. En effet, certains patrons possèdent entre eux une relation de précédence devant être respectée pour permettre une extension correcte et complète de la méthode considérée. Il est donc nécessaire de les organiser de façon spécifique dans la bibliothèque pour permettre de les exécuter dans un ordre qui soit cohérent. Cet ordre peut être obtenu par le biais des cartes d'extension.

Une carte d'extension est définie pour un domaine d'application spécifique, donc pour un ensemble précis de patrons d'extension. Elle est représentée par un graphe dont les nœuds sont les intentions que l'ingénieur de méthodes souhaite réaliser et les arcs les différentes manières de procéder à ces exécutions. Les patrons d'extension sont représentés dans les cartes par un ensemble de deux nœuds reliés par un arc (l'ingénieur de méthodes peut naviguer d'une intention à une autre en appliquant une certaine manière de faire, ce qui correspond à l'exécution d'un patron d'extension spécifique).

La Figure 19 illustre l'utilisation d'une carte d'extension par l'ingénieur de méthodes pour modifier sa méthode d'origine et obtenir une méthode étendue adaptée à la situation et à ses besoins.

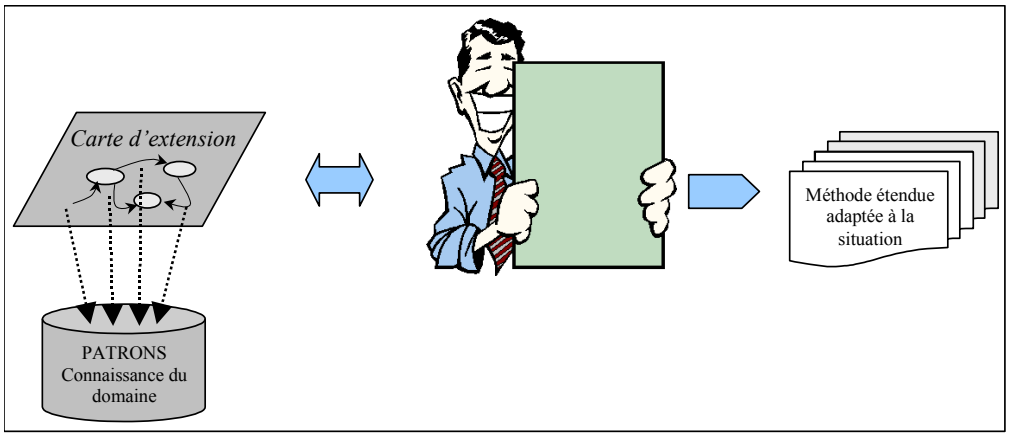


Figure 19: Processus d'organisation des patrons

Les patrons sont donc stockés dans une ***bibliothèque*** spécifique et l'approche utilise le ***processus d'organisation*** appelé *carte d'extension* pour les ordonner. Des règles de passage sont spécifiées pour permettre la création d'une carte d'extension particulière à partir d'un ensemble de patrons contenus dans la bibliothèque.

Cette solution peut donc se placer dans le cadre de l'organisation des approches de la façon suivante:

```
Organisation: bibliothèque, processus d'organisation
```

Les patrons d'extension peuplant la bibliothèque sont, dans la solution proposée, créés au fur et à mesure des besoins de l'ingénieur de méthodes. Pour simplifier le processus de construction de l'ingénieur, il existe donc une technique pour les créer de façon simple et systématique que nous avons appelée la *construction par méta-patrons*.

Les méta-patrons d'extension sont construits comme les patrons d'extension et ont sensiblement la même spécification. La différence fondamentale entre ces deux concepts est qu'un méta-patron s'instancie selon la méthode d'origine de l'ingénieur de méthodes pour définir un patron d'extension spécifique à cette méthode. La figure suivante illustre le processus de construction des patrons d'extension pour une méthode spécifique à l'aide des méta-patrons génériques prévus à cet effet.

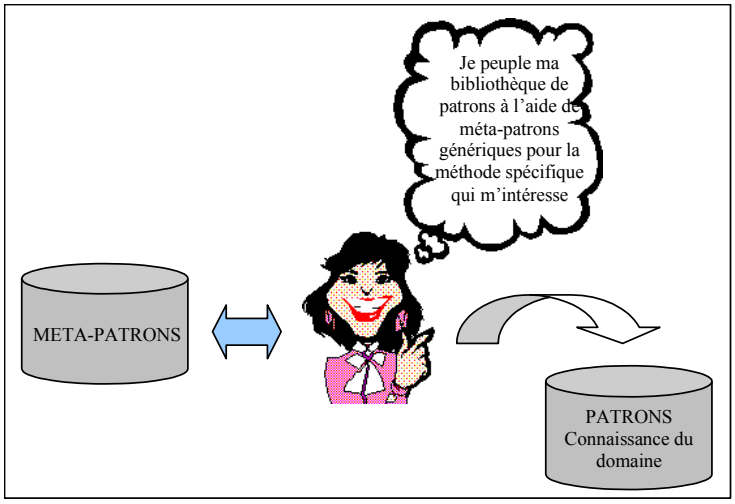


Figure 20: Processus de construction des patrons d'extension

L'approche proposée définit donc un processus ***formalisé*** de construction des patrons dans le but de créer ceux-ci d'une manière générique.

Cette solution peut donc se placer dans le cadre de la construction de la connaissance de la façon suivante:

```
Construction: formalisée
```

7 Récapitulatif de la solution proposée

La solution proposée par cette approche peut être résumée par la figure suivante.

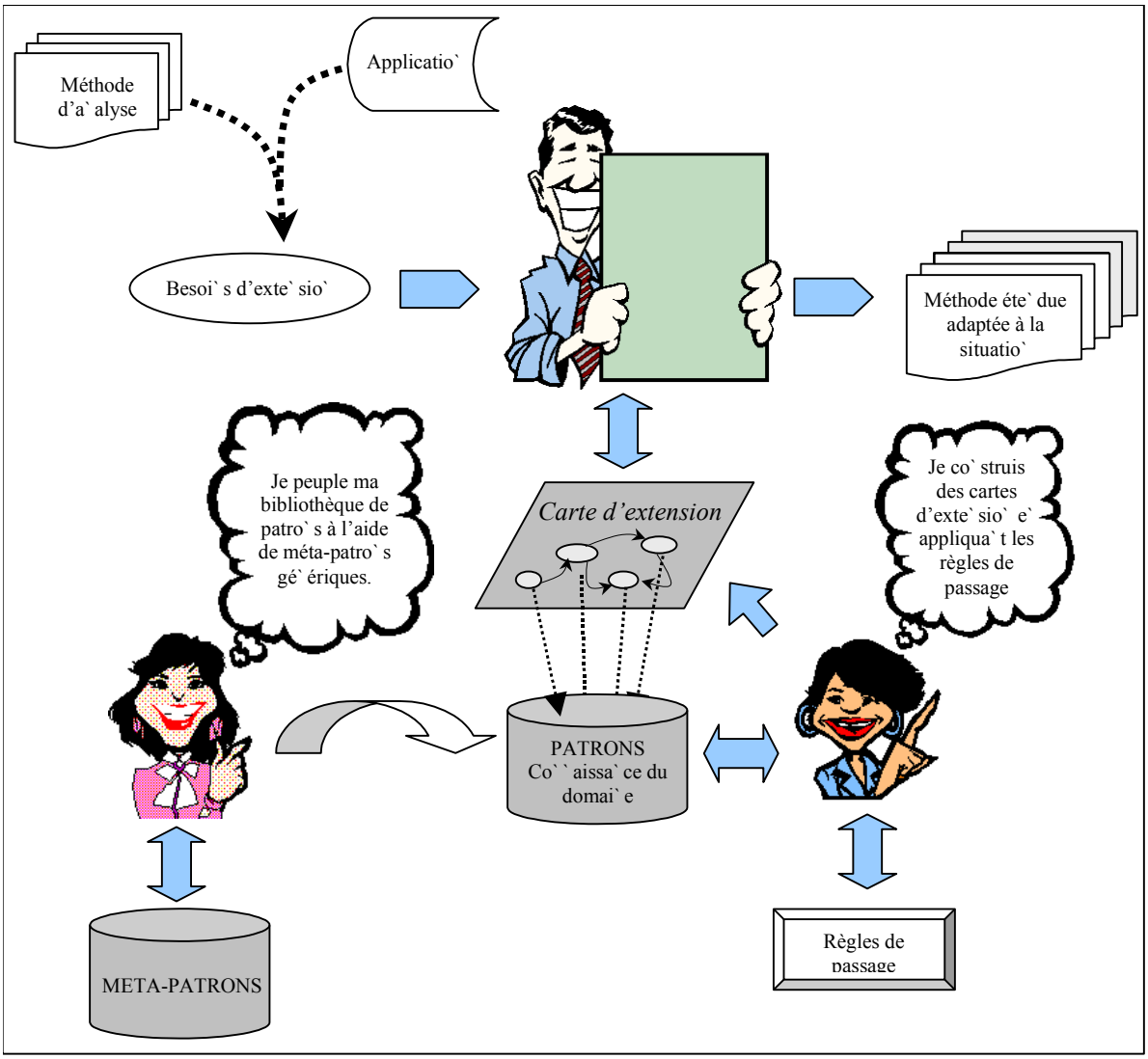


Figure 21 : Processus d'extension proposé dans ce mémoire

Cette solution peut donc se placer dans le cadre des approches existantes comme suit :

```
Nature : par contingence
Flexibilité : modulaire
Technique : instanciation, extension
Modèle : patron
Organisation :bibliothèque, processus d'organisation
Construction : formalisée
```

Chapitre IV : Description des patrons d'extension

Introduction

Il a été défini dans la section précédente que ce travail s'organisait dans le cadre des extensions de méthodes par l'utilisation de patrons génériques. Différentes définitions du terme de *patron* ont été données depuis quelques années.

- Pour Peter Coad [Coad92], un patron est *une forme entièrement réalisée, originale ou un modèle accepté ou proposé pour une imitation ; quelque chose qui est vu comme un exemple normatif pouvant être copié, archétypé ou utilisé comme exemple*; un patron orienté objet est *une abstraction d'un doublet, un triplet ou autre petit groupe de classes qui peut être utile encore et encore dans tout développement orienté objet.*
- D'aprés J. Rumbaugh, le concept de patron est une *tentative pour la représentation de l'expérience personnelle des développeurs de manière uniforme.*
- R. Johnson [Gamma93] se situe plus au niveau du développement : *un patron de conception identifie, nomme et abstrait des thèmes communs du processus de conception orienté objet.*
- Dans [ACM96] est introduite une définition plus générale : *un patron a pour but de décrire avec succès des solutions à des problèmes logiciels communs et d'aider les gens a réutiliser des pratiques vraies et résolues.*
- Pour C. Rolland [Rolland98], un patron est *plus qu'une description d'un élément du monde réel, c'est aussi une règle décrivant quand et comment créer cet élément. Le patron doit à la fois inclure une description de cet élément et une description de la démarche qui permet de générer cet élément*.

L'idée majeure concernant les patrons est que ceux-ci relient un problème à sa solution. Pour atteindre cet objectif, le patron doit comporter l'ensemble des propriétés suivantes [ACM96].

- La définition du *problème* pour lequel le patron est proposé (par ex. *« Besoin d'un référentiel temporel spécifique »*).
- La définition de sa *solution* (par ex. *« Comment créer un référentiel temporel spécifique »*). Cette solution s'appuie sur différents langages de description et de manipulation.
- La définition du *contexte d'utilisation* (par ex. *« La méthode n'offre pas la possibilité d'avoir un référentiel temporel spécifique »)*. Le contexte permet de spécifier quel est l'ensemble des situations où l'on peut appliquer le patron.

Différents formalismes de représentation intégrant ces trois aspects ont déjà été définis et sont globalement équivalents les uns par rapport aux autres. C. Alexander en a utilisé un pour décrire ses patrons nécessaires à la construction de bâtiments dans le domaine architectural. Ensuite, le formalisme utilisé par le *Gang of Four* [Gamma94] (patronyme regroupant E. Gamma, R. Helm, R. Johnson et J. Vlissides) utilise une structure composée de 14 rubriques. Il y a également P. Coad [Coad96] qui offre un ensemble de 148 stratégies et 31 patrons destinés à guider un concepteur dans la construction de modèles orientés objets effectifs et complets.

Le formalisme de représentation des patrons décrit dans ce mémoire a été défini de façon spécifique pour le problème de l'extension de méthodes orientés objet. Cette section décrit les particularités spécifiques des patrons d'extension utilisés.

8 Description des patrons

Le formalisme de représentation des patrons d'extension utilisé dans ce travail peut être représenté sous la forme du méta-modèle visualisé à la figure suivante.

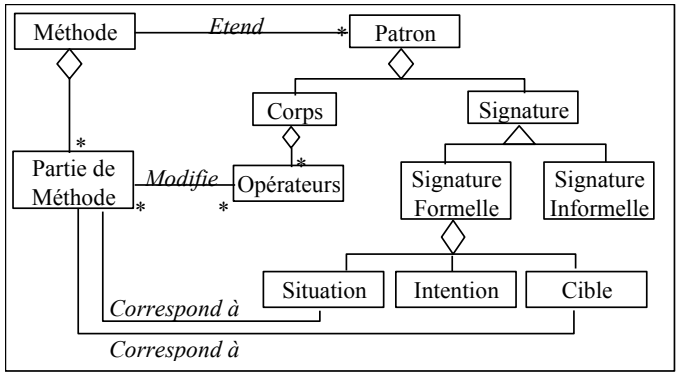


Figure 22: Méta-modèle d'un patron d'extension

On peut voir sur cette figure qu'un patron d'extension étend une méthode composée d'un ensemble de parties de méthode. La structure d'un patron peut se voir sous deux aspects : une partie représentant la connaissance réutilisable (corps) ainsi qu'une partie permettant de définir les aspects d'application de ce patron (signature). Le corps d'un patron d'extension est composé d'un ensemble d'opérateurs représentant les opérations de transformation à exécuter sur la méthode pour pouvoir l'étendre. La signature peut se voir sous deux aspects. La vue informelle de la signature permet de spécifier le domaine d'application, les heuristiques ainsi que les forces et les faiblesses du patron d'extension alors que la vue formelle permet de définir les parties de la méthode qui seront modifiées par le patron ainsi que l'extension spécifique que l'on souhaite effectuer. Tous ces aspects sont décrits en détail dans la suite de cette section.

8.1 Signature

La figure suivante montre la structure de la signature d'un patron. Chaque patron d'extension contient une partie, appelée *signature*, qui permet de définir ses aspects d'application. La signature d'un patron contient deux parties spécifiques : une partie informelle et une partie formelle. Cette dernière correspond à la notion d'*interface* de [Rolland98].

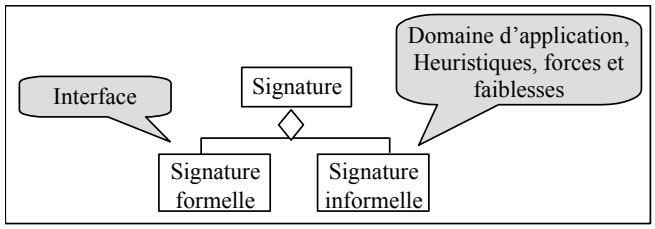


Figure 23: Structure de la signature d'un patron d'extension

8.1.1 Signature formelle (interface)

La signature formelle correspond à la notion d'interface permettant d'expliquer dans quelles situations et avec quelles intentions le patron est applicable. C'est donc un triplet <situation, intention, cible> associé au corps du patron, comme l'indique la Figure 24. On peut dire que l'interface est la partie visible du patron.

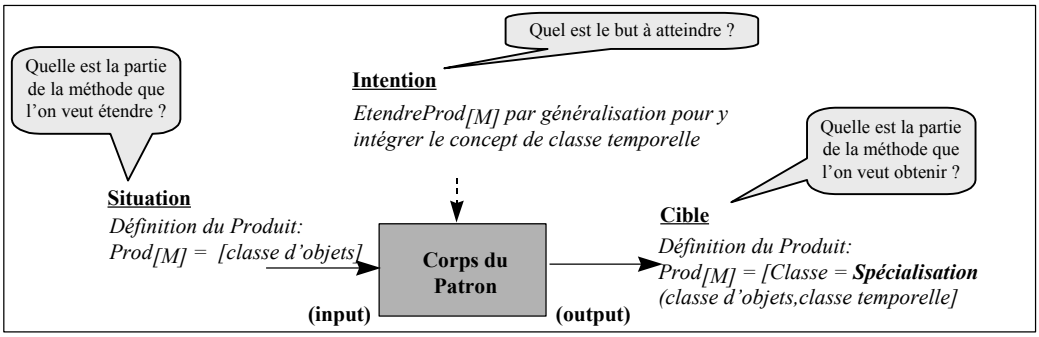


Figure 24 : La partie interface d'un patron d'extension

8.1.1.1 Situation

La *situation* est la partie de la méthode orientée objet à étendre. Elle représente toute partie du **PRODUIT** en cours de développement ou toute partie de la **DÉMARCHE** en cours d'exécution pouvant faire l'objet d'une prise de décision de la part de l'ingénieur d'application.

La granularité des situations est variable : celles-ci peuvent être *atomiques* comme, par exemple, lorsqu'elles font référence à un attribut unique ou à une seule classe, ou bien *complexes*, comme lorsqu'elles référencent une spécification entière ou un modèle dans son ensemble.

La situation consiste en la définition du **PRODUIT** ou de la **DÉMARCHE** à étendre dans le cas d'un patron d'extension. Elle peut être décrite en langage naturel ou avec les langages de description de la méthode (Sections 9.1.2 et 10.1.2).

Exemple de situation

Pour l'intention *« Etendre par Généralisation le concept de Classe Objet pour intégrer le concept de Classe Temporelle»*, la situation est représentée par la partie du **PRODUIT** concernant le concept de *Classe Objet* qui sera donc généralisé en *Classe* pour permettre de lui attribuer deux spécialisations: *Classe Objet* et *Classe Temporelle*.

8.1.1.2 Intention

L'intention reflète le choix que peut faire l'ingénieur d'application à un moment donné du processus. Elle référence l'extension, le but à atteindre pour utiliser le patron. Il peut s'agir d'un objectif très global comme celui *d'Etendre la méthode OMT* ou bien d'un but plus précis, par exemple d'*Etendre le concept de classe pour permettre la gestion d'historiques*. L'intention conduit le processus d'utilisation des patrons. La structure des intentions utilisée ici et représentée à la Figure 25 est un résumé de la structure définie dans [Prat99].

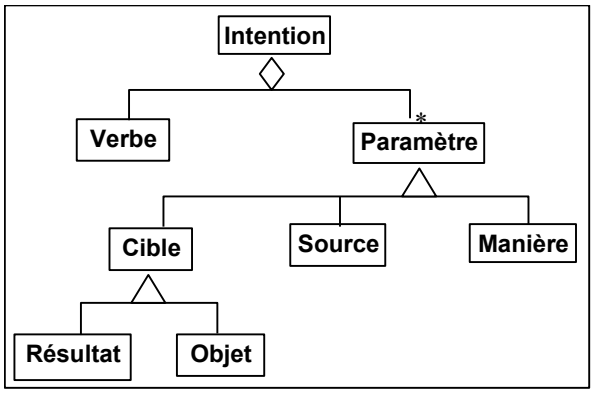


Figure 25: Structure d'une intention

Elle est décrite grâce au formalisme suivant qui la compose selon deux parties, ainsi que l'indique la Figure 25 :

- un verbe (Etendre, Définir...). Dans l'exemple *« Etendre, par Spécialisation Globale, Proc$_{[O^*]}$ pour y intégrer la construction des Classes Calendrier»*, le verbe est *Etendre*.
- un ensemble de paramètres. Il y a différents types de paramètres (source, cible, manière...). Chaque paramètre joue un rôle différent et a une signification dans l'intention exprimée.
    - La cible désigne ce qui va être modifié par l'exécution du patron. Il y a deux types de cibles, les objets et les résultats. Un objet est supposé exister avant la réalisation de l'intention, alors qu'un résultat est issu de la réalisation de l'intention. Dans l'exemple, la cible sera la partie de la **DÉMARCHE** représentant la construction d'une *Classe* et qui permettra de construire différents types de *Classes*, dont la *Classe Calendrier*, c'est donc un objet.
    - La source identifie l'origine de ce qui va être modifié par l'exécution. La source de l'exemple est la partie du **PRODUIT** concernant le concept de *Classe* et de *Classe Calendrier* ainsi que la partie de la **DÉMARCHE** concernant la *construction de la Classe.*
    - La manière exprime de quelle façon l'objectif va être atteint. Ici, la technique d'extension de la **DÉMARCHE** utilisée permet d'effectuer une extension Séquentielle Globale lorsque la partie **PRODUIT** de la méthode a été étendue par Spécialisation.

Exemple d'intention

Intention : *« Etendre, par Composition, le concept de Classe Objet pour y intégrer le concept de Contrainte Temporelle».*

Verbe	Paramètres		
	Source	**Cible**	**Manière**
Etendre	[Classe Objet = **Composition** (Propriété, Evénement]	[Classe Objet = **Composition** (Propriété, Evénement, Contrainte temporelle)]	Composition

8.1.1.3 Cible

La *cible* représente la partie de la méthode qu'il faut obtenir après application du patron.

Pour illustrer ce point, considérons l'interface (signature formelle) suivante : < *situation = ([Domaine]), intention = « Etendre, par Généralisation, le concept de Domaine non Temporel pour y intégrer le concept de Domaine Temporel»*,. La situation représente la méthode objet d'origine possédant le concept de *Domaine*. Le résultat de l'application (la cible) sera la même méthode objet mais incluant le concept de *Domaine Temporel*. Le corps du patron explique comment procéder pour remplir cette intention dans cette situation spécifique. Dans cet exemple, le corps du patron affine le concept de *Domaine* en définissant un nouveau type de domaine introduisant la notion de temps.

Exemple de cible

Interface : < *situation = ([Classe]), intention = « Etendre, par Spécialisation, le concept de Classe pour y intégrer le concept de Classe Temporelle»*.

Situation	Intention	Cible
Concept de *Classe*	Etendre, par Spécialisation, le concept de Classe pour y intégrer le concept de Classe Temporelle	Concept de *Classe Temporelle* comme concept spécialisé du concept de *Classe*

L'exécution de ce patron permet d'ajouter à la méthode d'origine la définition de la *Classe Calendrier*.

8.1.2 Signature informelle

La signature informelle permet de préciser certains détails concernant le patron d'extension, de façon textuelle et informelle.

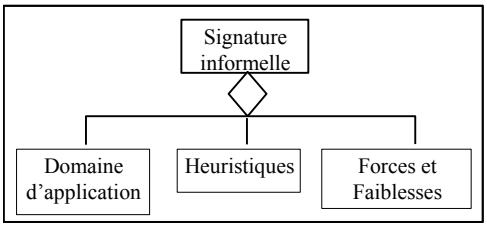


Figure 26: Structure de la signature informelle d'un patron d'extension

8.1.2.1 Domaine d'application

Un domaine est usuellement vu comme un champ d'application dans lequel des systèmes d'information existent déjà ou pourront être développés [Prieto-Diaz90]. On considère un domaine d'application comme une classe de problèmes pour lesquels il existe des solutions [Semmak98]. Dans le cadre de ce travail, nous prenons l'hypothèse qu'il est possible de s'intéresser à un domaine d'application à partir du moment où il existe, d'une part, un consensus établi par une communauté

d'ingénieurs autour d'un ensemble fini de problèmes (ou besoins), et d'autre part, une variété de solutions pour résoudre ces problèmes.

Exemples de domaine d'applications

Domaine	Besoins correspondants
Applications temporelles	Référentiels temporels, Historiques, Estampillages…
Applications multi-agents	Acteurs externes, Stimulus externes, Droits d'accès…

8.1.2.2 Heuristiques

Les *heuristiques* permettent à l'ingénieur de méthodes de savoir comment utiliser le patron pour satisfaire les besoins particuliers de l'application. Elles sont exprimées en langage naturel.

Exemple d'heuristique

Pour l'intention *« Etendre, par Spécialisation, le concept de Classe pour y intégrer le concept de Classe Calendrier»,* les heuristiques seront les suivantes : « La méthode d'origine contient un concept appelé *Classe* dont la définition est similaire au concept de *Classe Calendrier* que l'ingénieur de méthodes souhaite intégrer. La technique à utiliser sera donc de pratiquer une spécialisation dans le but d'affecter un autre type au concept généralisé de *Classe* qui sera la *Classe Calendrier*.

8.1.2.3 Forces et faiblesses

Cette partie permet de décrire à l'ingénieur de méthodes les avantages et inconvénients à appliquer ce patron d'extension. Comme pour les heuristiques, les *forces* et les *faiblesses* d'un patron d'extension sont écrites de manière informelle en langage naturel.

Exemple de forces et faiblesses

Prenons l'exemple de l'intention *« Etendre, par Généralisation, le concept de Domaine non Temporel pour y intégrer le concept de Domaine Temporel».* Les avantages à appliquer le patron d'extension correspondant à cette intention seront de permettre aux ingénieurs d'applications de donner à certains attributs une sémantique temporelle beaucoup plus forte que ce qu'autorisent les méthodes d'analyse habituelles. Cependant, ce patron d'extension ne peut s'appliquer que si la méthode possède une définition précise des référentiels temporels de l'application. Ceci n'est généralement pas le cas et l'ingénieur de méthodes devra, en conséquence, appliquer d'abord le patron d'extension permettant d'intégrer ce concept de référentiel temporel dans la méthode avant de pouvoir appliquer celui-ci, ce qui permettra de maintenir la cohérence de la méthode.

8.2 Corps

Contrairement à la partie signature d'un patron qui peut s'exprimer à la fois de façon formelle et de façon informelle, le corps d'un patron ne s'exprime que d'une manière formelle. Cette écriture se fait grâce aux langages de manipulation de méthodes décrits dans les sections 9.3 et 10.3. La solution proposée par un patron est une suite d'opérateurs à exécuter (d'opérations à effectuer) sur un produit spécifique pour permettre la modification désirée par l'extension.

Exemple de corps de patron

Prenons l'exemple de l'interface suivante : *« situation = (M [Classe = **Spécialisation** (Classe Acteur, Classe Objet)]), intention = Etendre, par Spécialisation, le concept de Classe pour y intégrer le concept de Classe Calendrier»,* l'opérateur à effectuer sur la méthode est la suivante.

```
Insérer-isa (Classe Calendrier, Classe)
```

Cet opérateur permet donc à l'ingénieur de méthodes d'intégrer un nouveau concept dans la méthode d'origine, le concept de *Classe Calendrier*, qui est une spécialisation du concept de *Classe* déjà présent dans la méthode. Ce concept est inséré avec toute sa définition et sa description.

8.3 Typologie des patrons d'extension

Nous avons défini précédemment qu'une méthode est composée de deux parties distinctes : la partie **PRODUIT** et la partie **DÉMARCHE**. Nous pouvons donc en déduire qu'il est donc nécessaire de différencier deux types de patrons d'extension : ceux qui étendront la partie **PRODUIT** et ceux qui étendront la partie **DÉMARCHE**. La Figure 27 illustre cette notion.

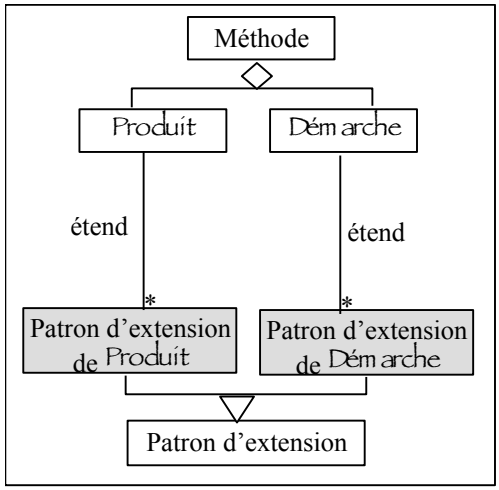


Figure 27: Typologie des patrons d'extension de méthodes

Comme les deux parties de la méthode sont fondamentalement différentes, les extensions réalisées par un patron d'extension de **PRODUIT** ou un patron d'extension de **DÉMARCHE** sont, elles aussi, spécifiques.

En effet, si le patron est un patron d'extension de **PRODUIT**, sa spécification ne concerne que la partie **PRODUIT** de la méthode d'origine. Elle permet de définir la situation avant et après extension du modèle de **PRODUIT** ainsi que les opérateurs de modification spécifiques pour aller de l'une à l'autre. De la même manière, si le patron est un patron d'extension de **DÉMARCHE**, sa spécification ne concerne que la partie **DÉMARCHE** de la méthode. Elle décrit la situation avant et après extension de la **DÉMARCHE** et les opérateurs de modification à effectuer. Il existe cependant une certaine dépendance entre ces deux types d'extension. En effet, la partie **DÉMARCHE** d'une méthode ne peut s'étendre que si sa partie **PRODUIT** a été étendue préalablement (il faut, par exemple, introduire les

concepts dans un premier temps avant de pouvoir insérer leurs étapes de construction dans un deuxième temps).

Les deux sections suivantes définissent la spécification de chacun de ces deux types de patrons d'extension.

9 Spécification d'un patron d'extension du PRODUIT d'une méthode

Nous avons défini, à la section 8.1.1, le fait que la signature formelle (interface) d'un patron était composée de quatre parties distinctes appelées la situation, l'intention, le corps et la cible. Comme l'indique la figure suivante, plusieurs formalismes ont été définis pour permettre une manipulation plus précise des patrons d'extension.

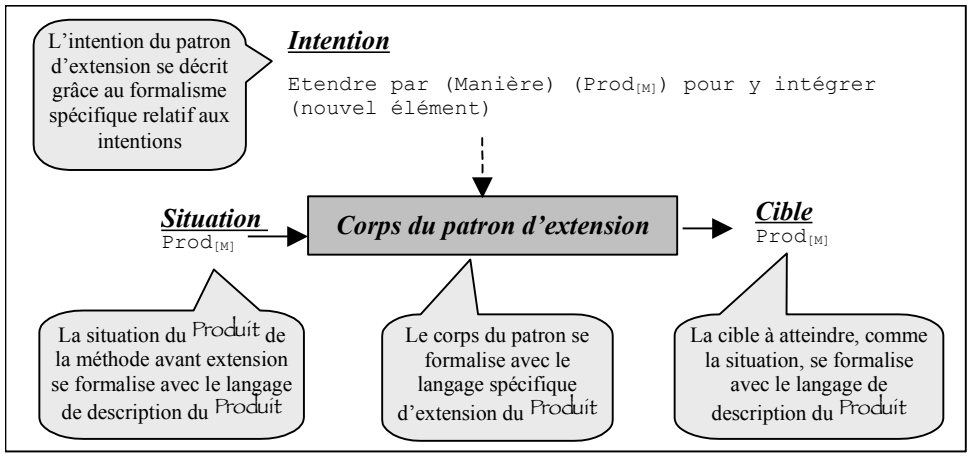


Figure 28: Interface d'un patron d'extension de Produit

On peut remarquer sur cette figure que les parties concernant la situation et la cible d'un patron d'extension du **PRODUIT** peuvent être décrites en utilisant le même formalisme de description puisqu'elles représentent toutes deux une partie du **PRODUIT** de la méthode. Ce langage de description est défini à la section suivante. La section 9.2 décrit la structure spécifique aux intentions des patrons d'extension de **PRODUIT** alors que la section 9.3 définit le langage de description des opérateurs de modification permettant de transformer le **PRODUIT** lors de l'extension et contenus dans le corps du patron.

9.1 Description de la partie Produit

La situation définit le **PRODUIT** *avant* extension et la cible *après* extension. Il est nécessaire de formaliser la description de la partie **PRODUIT** d'une méthode pour une extension plus facile et adaptable à toute méthode.

On peut voir à la Figure 28 que la situation et la cible sont représentées par l'expression $Prod_{[M]}$. Ce terme représente en fait la définition du **PRODUIT** ou partie du **PRODUIT** de la méthode d'origine (la méthode M).

Le formalisme utilisé pour cette définition est décomposé en deux parties distinctes : une pour la représentation graphique du **PRODUIT** et une pour la représentation textuelle correspondante. C'est principalement avec le langage textuel que les patrons d'extension sont définis mais il a été jugé utile de conserver une représentation graphique pour l'aide à la visualisation du **PRODUIT** de façon rapide et efficace.

9.1.1 Représentation graphique du Produit d'une méthode

La notation UML [Muller97] a été choisie pour la représentation graphique des parties de **PRODUIT** car elle est un langage de modélisation objet qui contient bon nombre de modèles et de concepts, intégrant la majorité des concepts des méthodes BOOCH [Booch94], OMT et OOSE [Jacobson93]. Elle consiste en un ensemble de sémantiques et de notations qui s'adressent à une large variété de domaines et de technologies d'implémentation. Elle peut même être utilisée dans des domaines auxquels ne s'intéressent généralement pas les méthodes existantes, comme les systèmes concurrents ou distribués. De plus, elle contient un ensemble de mécanismes d'extension (tels que les stéréotypes) qui permet d'adapter UML aux besoins des applications. Cette notation est devenue un standard de l'OMG et peut être utilisée indépendamment d'une méthode qui la met en œuvre.

Le modèle de représentation d'UML peut s'abstraire selon le schéma de la Figure 29. Selon la méthode utilisée, le terme de concept pourra représenter chacune des notions suivantes : classe d'objet, attribut, opération, contrainte ou lien. Ces concepts peuvent également être reliés entre eux selon trois types de liens : référence, composition et héritage.

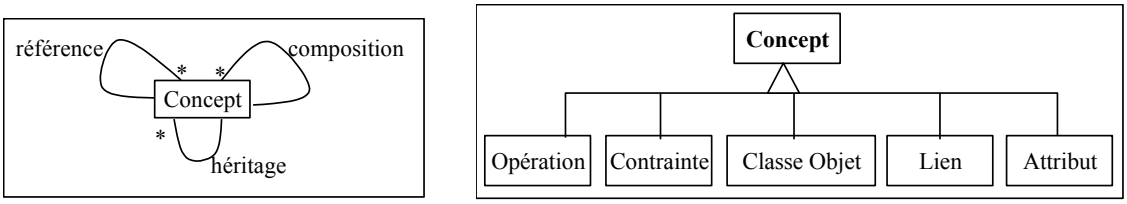


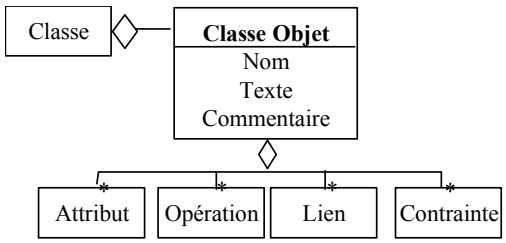
Figure 29 : Modèle générique de représentation du PRODUIT des méthodes orientées objet

9.1.2 Langage de description du Produit d'une méthode

La notation UML permet de représenter graphiquement tout **PRODUIT** de toute méthode d'analyse. Pour simplifier les étapes de modification dues aux extensions, un langage de description textuel a été formalisé.

Concept = ***Structure*** *({Attribut : TypeAttribut}),* ***Composition*** *({Concept}),* ***Spécialisation*** *({Concept}),*
Référence *({Concept}),* ***Compose*** *({Concept}),* ***Hérite-de*** *({Concept})*

Ce formalisme permet de spécifier les propriétés ainsi que les liens qu'un concept a avec les autres concepts (sa définition), c.-à-d. quels sont les concepts qui le composent, qui le spécialisent ou qu'il a en référence.



Classe Objet = **Structure** (Nom : char[10], Texte : char [20], Commentaire : char [50]), **Composition** ({Attribut} , {Opération} , {Lien} , {Contrainte}), **Hérite-de** (Classe)

Une classe d'objet est définie par son nom, un texte et un commentaire, mais est également une composition des concepts d'attributs, d'opérations, de liens et de contraintes. Une Classe Objet est également une spécialisation de Classe.

Figure 30 : Exemple de définition de concept

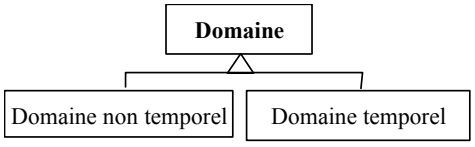
Comme le montre la Figure 30, la définition d'un concept porte donc à la fois sur sa structure (définition de ses attributs) et sur la définition des liens qu'il a avec son entourage (les autres concepts). Les attributs présents dans la structure du concept sont définis avec les domaines d'attributs simples usuels (chaîne de caractères, entier, date…).

{ Concept }

Cette écriture permet de formaliser la cardinalité multiple d'un concept par rapport à un autre concept (formaliser un ensemble). La Figure 30 illustre parfaitement cette notion d'ensemble. En effet, une classe est formée d'ensembles d'attributs, d'opérations, de liens et de contraintes.

Concept = ***Spécialisation*** *(Concept Spécialisé 1, Concept Spécialisé 2)*

Ce formalisme permet de spécifier quels concepts héritent d'un autre concept (spécialisation).



Domaine = **Spécialisation** (Domaine non Temporel, Domaine Temporel)

Un domaine peut se spécialiser en un domaine non temporel ou un domaine temporel.

Figure 31 : Exemple de spécialisation

Concept = Hérite-de (Concept1)

Ces termes permettent de représenter le lien inverse de la spécialisation. Par exemple, la notation *Domaine non Temporel = Hérite-de (Domaine)* permet de définir le fait que le concept de *Domaine non Temporel* est une spécialisation de celui de *Domaine*.

Concept = ***Composition*** *(Concept1, Concept2)*

Ce formalisme permet de spécifier quels concepts composent un autre concept. Un exemple de composition est décrit à la Figure 30.

Concept = Compose (Concept1)

Ces termes permettent de représenter le lien inverse de la composition. Par exemple, la notation *Contrainte = Compose (Classe)* permet de définir le fait que le concept de *Contrainte* fait partie de la composition de celui de *Classe*.

Concept1 = ***Référence*** *(Concept2)*

Ce formalisme permet de spécifier quels concepts référencent un autre concept.

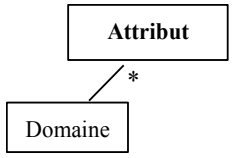
	Attribut = **Référence** (Domaine) Domaine = **Référence** (**{**Attribut**}**) Il y a une association entre le concept d'attribut et le concept de domaine avec une cardinalité (unaire, multiple) : un attribut a un et un seul domaine, le même domaine peut correspondre à plusieurs attributs.

Figure 32 : Exemple de référence

9.1.3 Exemples de descriptions de la situation et de la cible d'un patron d'extension de PRODUIT

Prenons par exemple pour situation un **PRODUIT** qui est représenté en UML par le schéma suivant.

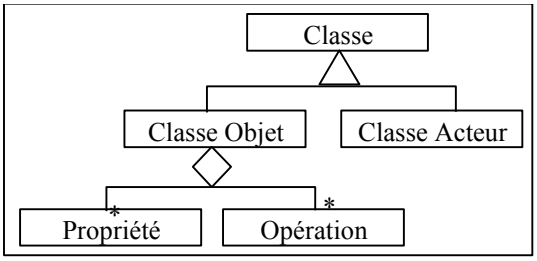


Figure 33: Exemple de situation de PRODUIT avec le formalisme d'UML

La manière de formaliser ce schéma avec le langage de description du **PRODUIT** sera donc la suivante.

```
Classe = Spécialisation (Classe Objet, Classe Acteur)
Classe Objet = Composition ({Propriété}, {Opération}}
```

Figure 34: Exemple correspondant avec le formalisme du langage de description du PRODUIT

Ce formalisme permet de spécifier quels sont les liens entre les concepts du modèle, ainsi que les cardinalités inhérentes à ceux-ci.

Prenons maintenant le cas de l'application, sur cette situation, d'un patron d'extension permettant d'intégrer le concept de *Contrainte* comme un nouveau composant de celui de *Classe Objet*. La représentation graphique de la cible obtenue après cette application sera donc celle que visualise la Figure 35.

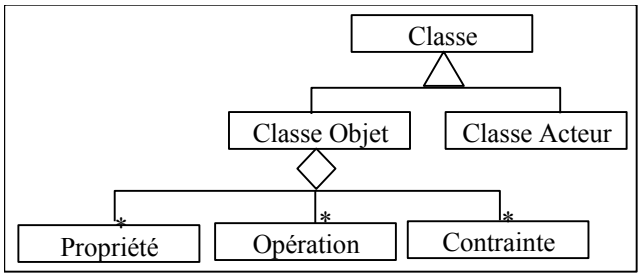


Figure 35: Exemple de cible de PRODUIT écrit avec le formalisme d'UML

La représentation textuelle associée à ce schéma est la suivante.

```
Classe = Spécialisation (Classe Objet, Classe Acteur)
Classe Objet = Composition ({Propriété}, {Opération}, {Contrainte})
```

Figure 36: Exemple écrit avec le formalisme du langage de description du PRODUIT

9.2 Structure des intentions des patrons d'extension de PRODUIT

L'intention permet de définir ce que l'ingénieur de méthodes veut obtenir après exécution du patron d'extension. Comme pour la description du **PRODUIT**, il est nécessaire de pouvoir utiliser un formalisme spécifique permettant une utilisation plus systématique des patrons.

9.2.1 Structure des intentions des patrons d'extension de PRODUIT

Le formalisme d'écriture des intentions a été indiqué à la section 8.1.1.2. Une intention est composée de plusieurs termes : un verbe et un ensemble de paramètres représentant la cible, la source ou encore la manière d'atteindre l'objectif désiré.

Cependant, la structure des intentions a également son importance et est définie de façon spécifique selon le type du patron d'extension (**PRODUIT** ou **DÉMARCHE**). Nous pouvons voir à la Figure 28 que l'intention d'un patron d'extension de **PRODUIT** a été définie de la façon suivante.

```
(Etendre)Verb par (Manière)Man (Prod[M])Obj pour y intégrer (nouvel élément)Res
```

Figure 37: Structure de l'intention d'un patron d'extension du PRODUIT

En effet, le *verbe* (verb) correspondant aux patrons d'extension du **PRODUIT** est toujours le verbe « Etendre ». La *manière* (Man) d'exécuter cette intention représente la façon d'effectuer l'extension du **PRODUIT**. L'*objet* (obj) concerné par l'extension est variable en fonction de la méthode d'origine et représente tout ou partie de son **PRODUIT**. Le *résultat* (res) correspond au nouvel élément que l'extension intègre dans cette méthode.

9.2.2 Exemple de la structure de l'intention d'un patron d'extension de PRODUIT

Prenons l'exemple de l'intention suivante.

(Etendre)$_{Verb}$ par (COMPOSITION)$_{Man}$ (le concept de Classe Objet)$_{Obj}$ pour y intégrer (le concept de Contrainte)$_{Res}$

Figure 38: Exemple de structure de l'intention d'un patron d'extension du PRODUIT

On peut déterminer dans cet exemple plusieurs points importants :

Ce que l'ingénieur de méthodes veut faire (le verbe), c'est *Etendre* sa méthode d'origine.

La façon dont on exécute cette extension (la manière), c'est par *Composition* (On compose un concept existant avec le concept que l'on souhaite intégrer dans la méthode).

Ce que le patron va modifier (l'Objet), c'est la partie du **PRODUIT** de la méthode correspondant à la situation, c'est à dire *le concept de Classe Objet*

Ce que l'on ajoute à cet objet (le résultat), c'est *le concept de Contrainte*.

9.3 Description des opérateurs de transformation du PRODUIT

Le corps d'un patron permet de définir les opérateurs à exécuter sur le **PRODUIT** de départ pour le modifier et pour qu'il réponde aux besoins définis dans l'intention. Ces opérateurs de transformation ont également été décrits selon un formalisme défini pour permettre une certaine généricité des extensions. Le formalisme de ces opérateurs est décrit ci-dessous.

9.3.1 Langage d'extension du PRODUIT d'une méthode

Renommer *(VieuxConcept, NouveauConcept)*

Cet opérateur particulier permet de changer le nom d'un concept tout en conservant sa définition et les liens qui lui sont attachés. Dans cet opérateur, les termes *VieuxConcept* et *NouveauConcept* représentent respectivement le concept dont on veut modifier le nom et celui que l'on veut lui attribuer. Par exemple, l'opérateur ***Renommer*** *(Domaine, Domaine non temporel)* permet de spécifier que le concept de *Domaine* de la méthode d'origine représente en fait le concept de domaine ne gérant pas de temps dans la méthode étendue.

Remplacer *(VieuxConcept, NouveauConcept)*

Il est parfois nécessaire de remplacer la définition d'un concept. Cet opérateur permet de remplacer un concept tout en conservant les liens que l'ancien élément avait avec le reste du modèle. Le terme *VieuxConcept* représente le concept que l'on souhaite remplacer alors que celui de *NouveauConcept* représente le concept que l'on intègre à sa place. Par exemple, l'opérateur ***Remplacer*** *(Classe Horloge, Classe Calendrier)* permet de spécifier le fait que le concept de Classe Horloge de la méthode de base n'a pas une définition correcte par rapport à l'extension que l'ingénieur de méthodes veut appliquer et qu'il est nécessaire de le remplacer. Cependant, les liens existants entre ce concept incorrect et les autres éléments de **PRODUIT** doivent être conservés.

Insérer *(NouveauConcept)*

Il est courant de devoir ajouter des concepts à la méthode lors d'une extension de celle-ci. Cet opérateur permet d'insérer un nouveau concept comme, par ex. l'opération ***Insérer*** *(Domaine)* qui permettra d'insérer dans le **PRODUIT** de la méthode un nouveau concept représentant le domaine. Il est à noter que l'insertion d'un nouveau concept insère à la fois le concept et toute sa spécification. En effet, si ce concept possède des spécialisations ou encore s'il est un composé d'autres concepts, alors ces concepts également seront insérés dans la méthode. Par exemple, si l'on prend l'opérateur ***Insérer*** *(Domaine Temporel),* les insertions concerneront à la fois le concept de *Domaine Temporel* mais également les concepts spécialisés de celui-ci, c'est à dire *Domaine Temporel Instant*, *Domaine Temporel Période* et *Domaine Temporel Intervalle*.

Supprimer *(VieuxConcept)*

Il est parfois nécessaire de supprimer un concept lors de l'extension du **PRODUIT**. Si l'on prend l'exemple d'une extension temporelle du concept d'attribut, dans une méthode contenant le domaine *Date*, l'extension utilisera l'opérateur ***Supprimer*** *(Date)* pour supprimer ce concept particulier, ce qui permettra de le spécialiser plus profondément avec différents domaines temporels.

Supprimer-isa *(ConceptSpécialisé, ConceptGénéralisé)*

Il arrive que la définition d'un concept soit correcte mais que ses liens avec le reste du modèle ne le soient pas. Dans ce cas, il sera utile de pouvoir supprimer les liens incorrects. Cet opérateur particulier permet de supprimer un lien de Spécialisation. Par exemple ***Supprimer-isa*** *(Classe estampillée temps validité, Classe)* supprimera le lien entre le concept de Classe et le concept de Classe estampillée avec un temps de Validité, ce qui permettra à l'ingénieur de méthodes de modifier les généralisations concernant les classes temporelles de la méthode.

Supprimer-comp *(ConceptComposé , ConceptComposant)*

Pour la même raison, il est parfois utile de supprimer un lien de composition entre deux concepts.

Supprimer-ref *(ConceptRéférencé , ConceptRéférençant)*

De la même manière, il est possible de supprimer un lien de référence entre deux concepts.

Insérer-isa *(ConceptSpécialisé , ConceptGénéralisé)*

Les extensions permettent souvent de spécialiser un concept selon les besoins de l'application. Dans le cas d'une extension temporelle du concept d'attribut, l'opérateur ***Insérer-isa*** *(Domaine non temporel, Domaine)* permettra d'insérer un lien d'héritage entre le concept spécialisé de Domaine non temporel et le concept de Domaine.

Insérer Cluster *(ConceptSpécialisé1, ConceptSpécialisé2 … , ConceptSpécialiséN)*

De la même façon, il est possible d'insérer un cluster entre plusieurs concepts spécialisés. L'opérateur ***Insérer Cluster*** *(Domaine temporel, Domaine non temporel)* permet de définir qu'un domaine doit être soit temporel, soit non temporel.

Insérer-ref *(Concept1 , Concept2 , rôle1, rôle2, cardinalité1, cardinalité2)*

Comme pour les liens d'héritage, il est important d'avoir la possibilité d'insérer des liens de référence entre les concepts. Par exemple, l'opérateur ***Insérer-ref*** *(Domaine temporel* **,** *granule, « comprend », « référence », un, plusieurs)* permettra de lier le concept de *Domaine Temporel* et celui de *Granule* avec un lien de référence ayant une cardinalité unaire d'un côté (la définition d'un domaine temporel ne comprend qu'un et un seul granule), et multiple de l'autre, (un granule peut correspondre à la définition de plusieurs domaines temporels).

Insérer-comp *(Concept1 , Concept2 , cardinalité)*

Pour finir, comme pour les liens d'héritage et de référence, il est important d'avoir la possibilité d'insérer des liens de composition entre des concepts déjà existants. L'opérateur ***Insérer-comp*** *(Opération , Classe Objet , plusieurs)* permettra de composer le concept de classe d'objets avec celui d'opération et ceci avec une cardinalité multiple (une classe d'objets est composée d'un ensemble d'opérations).

Généraliser *(Définition, ConceptSpécialisé , ConceptGénéralisé)*

Ce formalisme donne la possibilité à l'ingénieur de méthodes de généraliser tout ou partie de la définition d'un concept spécialisé vers son concept généralisé. Par exemple, l'opérateur ***Généraliser*** *(**Structure** (nom : char[10], texte : char[20],* ***Composition*** *({Propriété}, {Opération}), Classe Objet, Classe)* permet de généraliser une partie de la définition du concept de *Classe Objet* vers le concept de *Classe*.

9.3.2 Exemple de description des opérateurs de transformation d'un PRODUIT

Prenons l'exemple d'une partie de la partie **PRODUIT** de la méthode OMT représentant un concept composé d'autres concepts. La figure suivante décrit cette situation par le biais du langage de description du **PRODUIT**. Dans cet exemple, le concept de *Classe Objet* est composé de deux autres concepts : ceux de *Propriété* et d'*Opération*.

```
Classe Objet = Composition ({Propriété}, {Opération})
```

Figure 39: Description d'une partie du PRODUIT de OMT

Modifions maintenant ce **PRODUIT** pour introduire un nouveau concept dans la composition de la *Classe Objet* : le concept de *Contrainte*. La figure suivante illustre les opérateurs à exécuter sur le **PRODUIT** pour opérer cette modification avec le langage de manipulation du **PRODUIT**.

```
Insérer (Contrainte)
Insérer-comp (Contrainte, Classe Objet)
```

Figure 40: Modification du Produit de OMT

La partie de **PRODUIT** correspondant au résultat de cette opération de modification est illustrée dans la figure suivante.

```
Classe Objet = Composition ({Propriété}, {Opération}, {Contrainte})
```

Figure 41: Description de la partie du PRODUIT obtenu de OMT

9.4 Exemple de l'interface d'un patron d'extension de PRODUIT

Le processus d'extension correspondant à l'exemple utilisé dans cette section peut donc être illustré par la figure suivante. Cette figure visualise l'interface du patron d'extension correspondant.

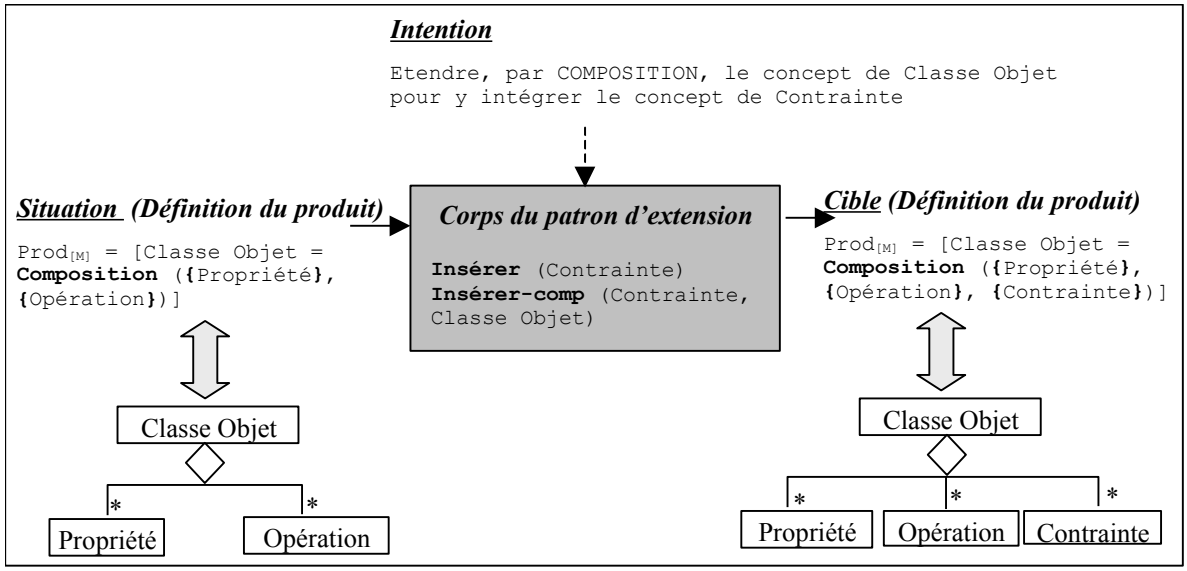


Figure 42: Exemple d'exécution du corps du patron d'extension

9.5 Exemple d'application d'une extension de PRODUIT d'une méthode

9.5.1 Problème

Prenons l'exemple de la méthode O* et d'un ingénieur de méthodes ayant besoin, entre autres, de la possibilité de définir plusieurs référentiels temporels.

9.5.2 Méta-modèle de la partie PRODUIT de la méthode d'origine

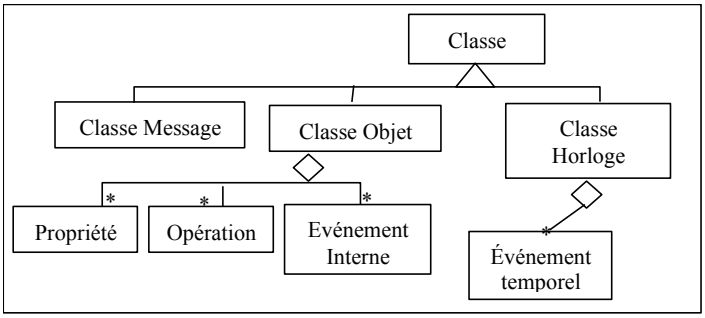


Figure 43: Partie du méta-modèle de la méthode O*

La méthode O* contient trois types de classes appelées *Classe Objet*, *Classe Message* et *Classe Horloge*. La *Classe Objet* décrit et regroupe un ensemble d'objets de même nature en terme de structure et de comportement. La *Classe Message* définit la description des messages. La *Classe Horloge* peut être assimilée à l'horloge interne du système et ne possède qu'une seule occurrence qui correspond au *Calendrier Grégorien*. La déclaration de la *Classe Objet* définit d'autres concepts comme les *Propriétés*, les *Opérations* et les *Evénements*.

La méthode O* possède donc le concept de *Classe Horloge* mais celui-ci n'est défini que pour le *Calendrier Grégorien*. Ce concept ne permet donc pas à l'ingénieur de méthodes de définir plusieurs référentiels temporels, comme il le désirerait. Cependant, il a la possibilité d'exécuter un patron d'extension permettant d'étendre la méthode en remplaçant ce concept par un concept de *Classe Calendrier* qui correspond plus à ce qu'il recherche. Malgré tout, comme les liens présents dans le méta-modèle et qui relient le concept de *Classe Horloge* au reste du modèle conviennent, il ne sera pas nécessaire de les supprimer. L'opérateur d'extension à appliquer sera donc celui permettant d'effectuer un remplacement de concept.

9.5.3 Description du patron d'extension

L'interface du patron d'extension d'une méthode au référentiel temporel est illustrée par la Figure 44 suivante.

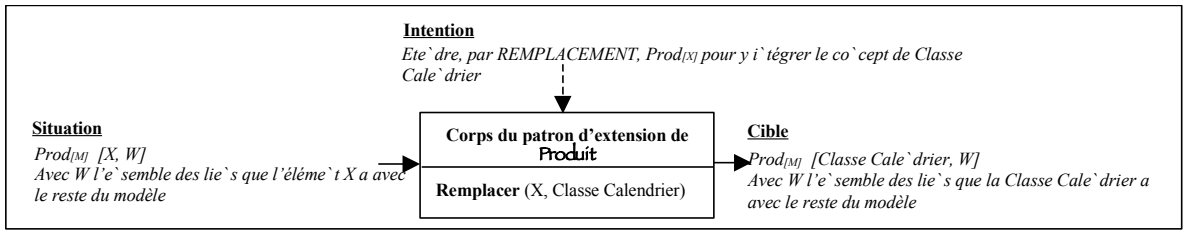


Figure 44: Interface d'un patron d'extension d'une méthode au référentiel temporel

Signature formelle

On peut constater sur cette figure que l'intention du patron d'extension est bien d'étendre la méthode d'origine en effectuant un remplacement d'un de ces concepts avec celui de Classe Calendrier. La situation du patron est donc le concept à supprimer – *X* - ainsi que l'ensemble de ses liens - *W* - alors que la cible est composée du nouveau concept - *Classe Calendrier* - et du même ensemble de liens - *W*.

Corps

L'opérateur de modification à effectuer sur la méthode d'origine sera donc ***Remplacer*** *(X, Classe Calendrier)* où *X* représente le concept de la méthode d'origine que l'on souhaite remplacer.

Signature informelle

Le domaine d'application de ce patron est celui des applications temporelles. Les heuristiques qui lui sont associées sont les suivantes.

La méthode d'origine contient un concept similaire au concept de Classe Calendrier mais celui-ci ne possède pas la définition et la description qui permettent de définir les référentiels temporels tels que le souhaite l'ingénieur de méthodes. Cependant, tous les liens qui rattachent ce concept au reste du modèle correspondent et peuvent donc être conservés. La technique à utiliser pour étendre la méthode d'origine est donc une technique de remplacement : l'ancien concept est supprimé et remplacé par le nouveau concept de Classe Calendrier alors que tous les liens de l'ancien concept sont conservés pour le nouveau.

Les forces et les faiblesses de ce patron sont décrites de la manière suivante.

Le point fort de l'application de ce patron est de permettre à l'ingénieur de méthodes la manipulation de plusieurs référentiels temporels, ce qui est utile dans les cas où une application est partagée par plusieurs entités n'ayant pas le même référentiel temporel (département d'une entreprise, pays). Par contre, le fait de manipuler plusieurs référentiels oblige les développeurs de l'application à spécifier systématiquement quel est le référentiel auquel ils se réfèrent lors de leurs actions.

9.5.4 Instanciation du patron d'extension pour la méthode O*

L'interface de cet exemple est illustrée à la Figure 45 suivante.

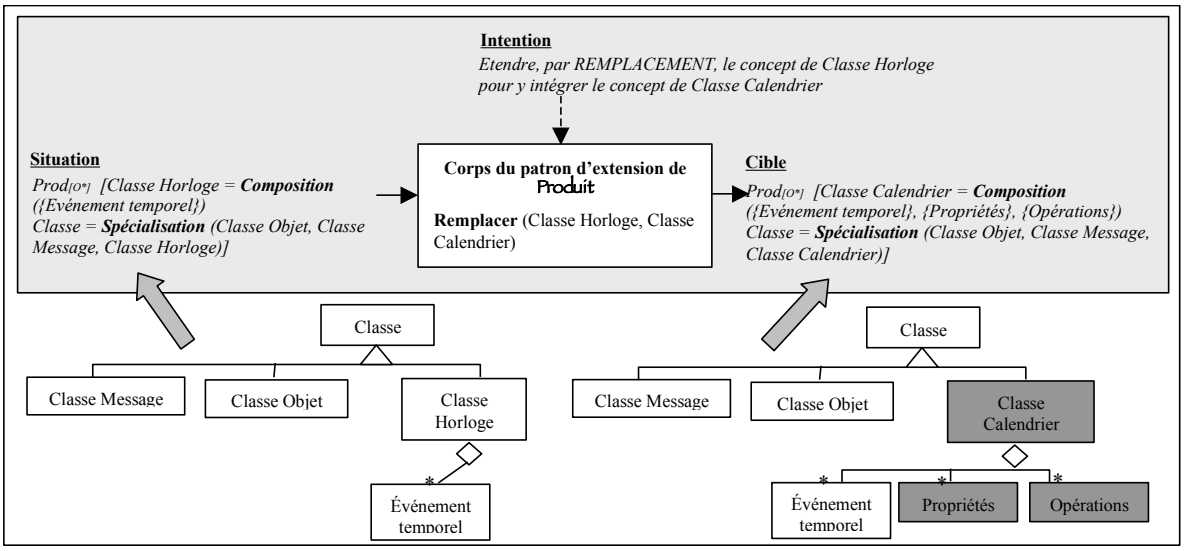


Figure 45: Application du patron d'extension de la méthode O* permettant de remplacer le concept de référentiel temporel

L'interface de cet exemple définit trois aspects importants du patron d'extension appliqué à la méthode O*.

La situation du **PRODUIT** de la méthode O* lors de l'application du patron ($Prod_{[O*]}$) est donc la suivante :

```
Classe Horloge = Composition ({Evénement temporel})
Classe = Spécialisation (Classe Objet, Classe Message, Classe Horloge)
```

L'intention du patron d'extension est :

```
Etendre, par REMPLACEMENT, le concept de Classe Horloge pour y intégrer le
concept de Classe Calendrier
```

La situation du **PRODUIT** obtenue après extension ($Prod_{[O^* \text{ etendu}]}$)est :

```
Classe Calendrier = Composition ({Evénement temporel}, {Propriétés},
{Opérations})
Classe = Spécialisation (Classe Objet, Classe Message, Classe Calendrier)
```

Le corps du patron est composé d'une seule opération permettant le remplacement du concept précité.

```
Remplacer (Classe Horloge, Classe Calendrier)
```

9.5.5 Méta-modèle de la partie PRODUIT de la méthode O* étendue

On peut voir sur la Figure 46 que la partie **PRODUIT** de la méthode O* a été étendue pour intégrer une nouvelle définition du concept de *Classe Calendrier* afin de permettre à l'ingénieur de méthodes de définir plusieurs référentiels temporels.

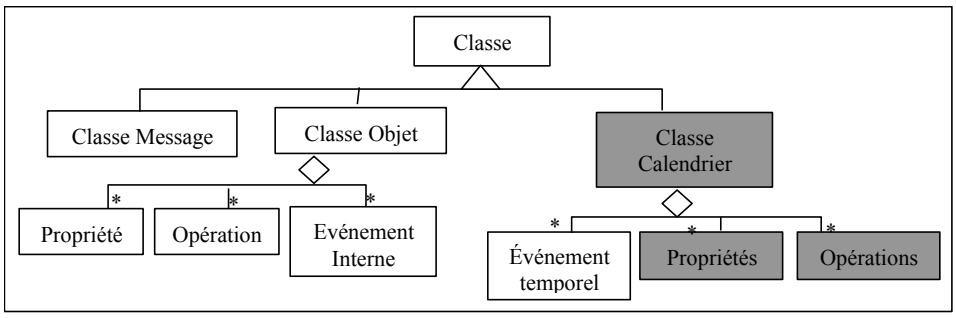


Figure 46: Partie du méta-modèle de la méthode O* étendue

L'application de ce patron d'extension a donc fourni le résultat escompté par l'ingénieur de méthodes et lui a permis de répondre au besoin exprimé.

10 Spécification d'un patron d'extension de la DÉMARCHE d'une méthode

L'interface d'un patron d'extension de **DÉMARCHE**, comme celle d'un patron d'extension de **PRODUIT**, est composée de quatre parties distinctes : la situation, l'intention, le corps et la cible. La figure suivante illustre les spécificités inhérentes aux interfaces des patrons d'extension de la **DÉMARCHE** d'une méthode.

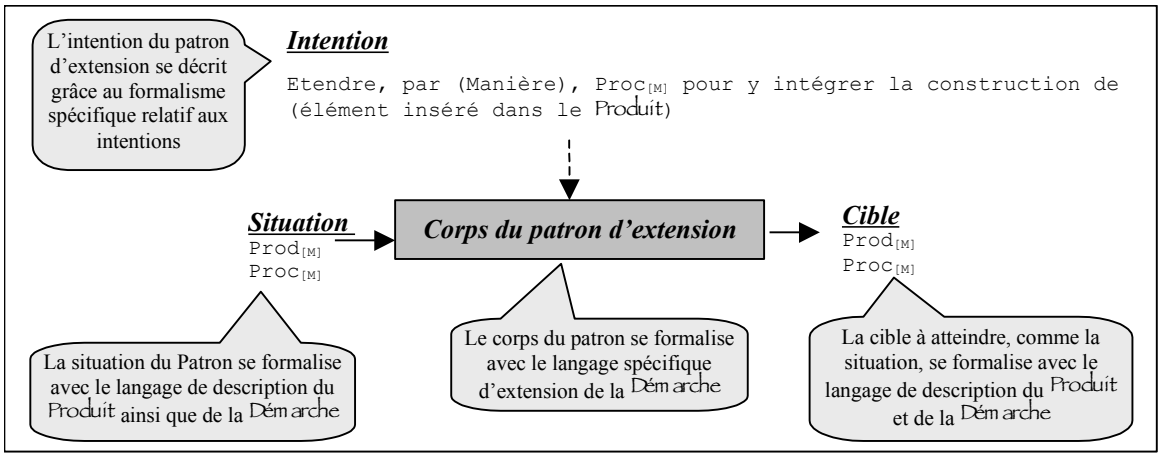


Figure 47: Interface d'un patron d'extension de DÉMARCHE

La situation et la cible d'un patron d'extension de la **DÉMARCHE** d'une méthode sont toutes les deux composées de deux éléments distincts : l'un concernant la partie **PRODUIT** et l'autre la partie **DÉMARCHE**. En ce qui concerne la description de la partie **PRODUIT**, les formalismes de description ont déjà été définis dans la section 9.1. La description de la partie **DÉMARCHE** est indiquée dans le paragraphe suivant. La structure de l'intention d'un patron d'extension de **DÉMARCHE** est expliquée ensuite dans la partie 10.2 qui sera suivie de celle décrivant le formalisme utilisé dans les corps de ces patrons et appelé le langage d'extension de la **DÉMARCHE** d'une méthode.

10.1 Description de la partie DÉMARCHE

Lors d'une extension de la partie **DÉMARCHE** d'une méthode, l'ingénieur de méthodes a besoin de décrire à la fois la partie **PRODUIT** et la partie **DÉMARCHE** de celle-ci. La description de la partie **PRODUIT** a été détaillée dans la section 9.1., la partie **DÉMARCHE** est décrite grâce au langage de description défini dans cette section.

On peut voir à la Figure 47 que la situation, comme la cible, est représentée par les expressions $Prod_{[M]}$ et $Proc_{[M]}$. Ces termes représentent en fait la définition du **PRODUIT** ou partie du **PRODUIT** de la méthode d'origine (la méthode M) ainsi que la définition de sa **DÉMARCHE** ou partie de sa **DÉMARCHE**.

De la même manière que pour la spécification des patrons d'extension de **PRODUIT**, la description de la partie **DÉMARCHE** est composée de deux parties distinctes : une représentation graphique et une représentation textuelle. Cette dernière est utilisée pour permettre une écriture plus formelle de la **DÉMARCHE** mais la représentation graphique a été conservée pour aider à une visualisation plus simple de celle-ci.

10.1.1 Représentation graphique de la DÉMARCHE d'une méthode à étendre (N, TURE)

Le formalisme que nous avons choisi vient du projet Esprit « NATURE[8] » décrit dans [Rolland95], [Rolland96], [Jarke99].

Le formalisme de modélisation NATURE [Rolland94], [Rolland95] consiste en un ensemble de concepts génériques et de leurs relations permettant de construire différents modèles de processus d'une façon structurée et détaillée. Cet ensemble fonctionne comme une coquille à partir de laquelle les modèles de processus centrés guidage peuvent être générés par instanciation puis exécutés pour guider le développement des différents projets d'ingénierie des besoins.

L'approche de modélisation de processus NATURE couple le contexte d'une intention à l'intention elle-même. Cette approche tend à définir non seulement les activités mais aussi pourquoi elles sont exécutées (les intentions) et quand (leurs contextes).

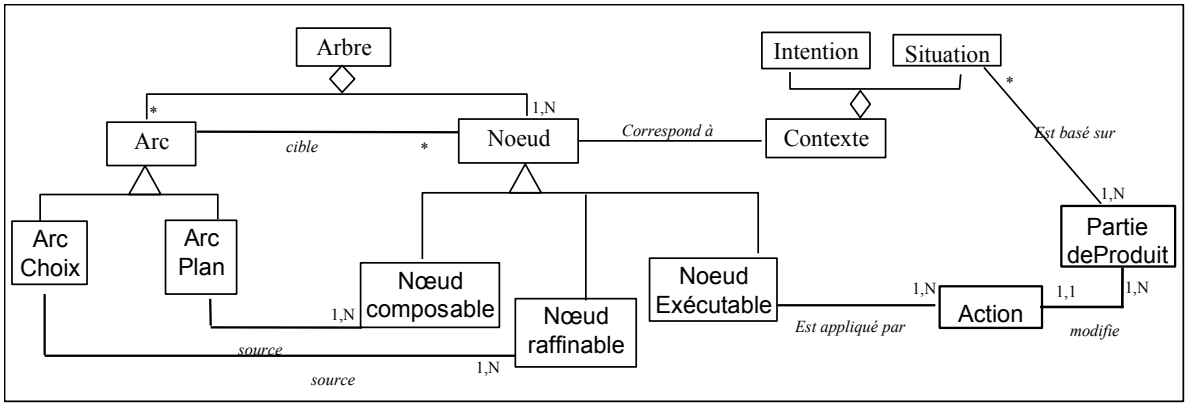


Figure 48 : Une version révisée du méta-modèle de processus NATURE

Le formalisme est présenté en détail dans [Rolland95] et [Jarke99].

L'aspect central d'un processus est la notion de *situation* reliée à la gestion de *contextes*. La Figure 48 donne un aperçu du modèle de processus, introduisant les concepts clés et leurs relations.

- Le concept central de ce méta-modèle est celui de contexte. Un contexte permet de décrire une étape de processus. Dans une situation, le concepteur doit avoir une intention pour progresser dans le processus RE, le contexte est donc un couple <situation, intention>. Un contexte permettant aux développeurs d'étendre un attribut est décrit de la manière suivante : <*Attribut, Etendre Attribut* > où A*ttribut* est la situation et *Etendre* l'intention.
- Une situation est le plus souvent une partie de la spécification qui permet d'avoir une intention. Les situations sont construites à partir des parties de la spécification pendant le processus RE.

[8] Projet Esprit n°6353. NATURE est un acronyme pour « Novel Approaches to Theories Underlying Requirements Engineering ».

- Une intention reflète un choix que l'ingénieur fait à un certain moment dans le processus RE. Une intention exprime ce que l'ingénieur veut atteindre (c'est un but) mais encapsule également deux autres aspects : la cible et l'approche. Une cible est la partie de la spécification que l'ingénieur veut atteindre. Une approche caractérise le moyen de remplir une intention.

Les contextes peuvent être reliés d'une façon hiérarchique pour définir des arbres. Un arbre est composé de nœuds (correspondant chacun à un contexte particulier) et d'arcs. Comme le montre la Figure 48, les arcs peuvent être de deux types: les arcs de choix qui permettent de raffiner un nœud en sous-nœuds et les arcs de plan qui permettent de décomposer un nœud en sous-nœuds. Le corps du contexte correspondant à chaque nœud permet de décrire ce que l'analyste doit faire pour opérationaliser sa décision.

Il y a trois types de contextes : le contexte exécutable, le contexte choix et le contexte plan, organisés en arbre de contextes, c'est à dire qu'ils seront graphiquement représentés par un nœud composable, un nœud raffinable ou un nœud exécutable.

Contexte exécutable

Au niveau le plus détaillé, l'exécution d'un processus RE peut être vue comme un ensemble de modifications exécutées sur le **PRODUIT** en cours de développement, chaque modification résultant de l'exécution d'une action déterministe qui fait partie de la description du contexte. Une telle action est la conséquence d'une intention faite dans une certaine situation. Ce type de contexte est appelé exécutable. Par exemple, le contexte <*Attribut, Définir Domaine de l'attribut comme Entier*> est un contexte exécutable car on ne peut ni le raffiner ni le composer. Un contexte exécutable implémente une intention, et qui est réalisée par une action. Il est à noter qu'exécuter une action change le **PRODUIT** en construction et peut générer une nouvelle situation qui sera elle-même sujette à de nouvelles intentions.

Contexte choix

Lorsqu'il crée une spécification, l'ingénieur d'applications peut faire face à plusieurs choix pour concrétiser son intention. Il doit donc sélectionner le plus approprié. Pour modéliser une telle partie de la connaissance du processus, nous introduisons une seconde spécialisation du concept de contexte, le contexte de choix. L'exécution d'un tel contexte consiste à choisir l'une de ses possibilités, c.-à-d. sélectionner un contexte représentant une stratégie particulière pour la résolution du contexte. Pour aider au processus de sélection, des arguments et des critères de choix sont introduits. Un choix permet de progresser dans le processus par raffinement. Par exemple, le contexte < *Description du Problème, Identifier Classe*> est un contexte choix car il peut être raffiné en deux autres contextes < *Description du Problème, Identifier Classe Objet*> et < *Description du Problème, Identifier Classe Acteur* > (cf Figure 49). Une alternative d'un contexte choix peut être de n'importe quel type, c.-à-d. exécutable, choix ou plan.

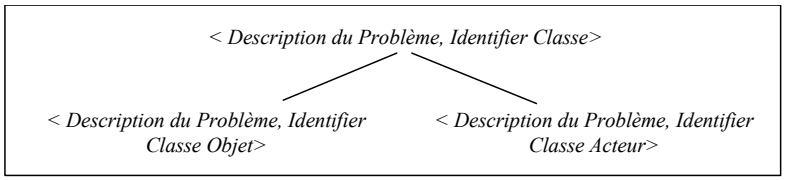


Figure 49 : Exemple de représentation graphique d'un contexte choix.

Contexte plan

Pour représenter les situations requérant un ensemble d'intentions, le méta-modèle inclut un troisième type de contexte appelé contexte plan. Comme il permet de décrire une séquence d'intentions à prendre, ce type de contexte peut être vu comme représentant une macro décision décomposable en plusieurs sous-intentions. Par exemple, le contexte <*Classe, Décrire Classe*> est un contexte plan car il peut être décomposé en trois sous contextes : <*Classe, Définir Attributs*>, <*Classe, Définir Opérations*>, <*Classe, Définir Contraintes*>.

Le chemin à prendre parmi ces sous-intentions est décrit dans un graphe de transitions. Chaque contexte plan en possède un. Les nœuds représentent les contextes composants du plan et les liens (les liens de précédence) les transitions possibles d'un contexte à un autre (soit leur ordonnancement, soit leur parallélisme), ces transitions pouvant être sujettes à des conditions spécifiées pour chacune, selon la situation, ce qui permet à l'analyste de trouver le contexte suivant à appliquer lorsqu'il suit le chemin décrit dans le contexte plan.

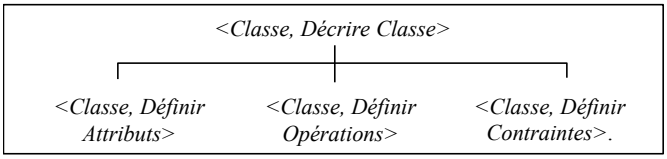


Figure 50 : Exemple de représentation graphique d'un contexte plan

Arbre de contextes

Chaque type de contexte influence le processus d'exécution de différentes manières : un contexte exécutable affecte le produit en cours de développement et génère une nouvelle situation pouvant elle-même devenir le sujet de certaines décisions ; un contexte choix ne change pas le produit mais aide à raffiner une intention ; un contexte plan donne le moyen de gérer la complexité d'une intention grâce à un mécanisme de décomposition.

Le processus est décomposé en plusieurs contextes organisés en arbre, ce qui permet de gérer le niveau d'abstraction de la définition du processus. Cet arbre permet de guider l'analyste des règles du plus haut niveau à celles du plus bas niveau.

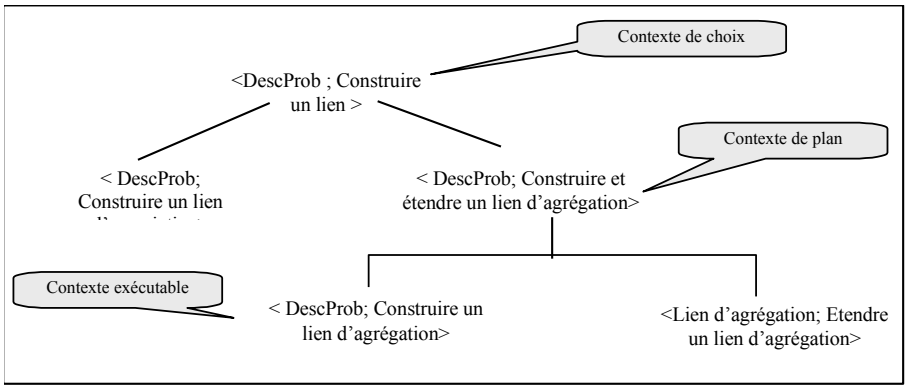


Figure 51 : Exemple d'un arbre de contextes

Cette représentation graphique est associée à un langage de description textuel, permettant de formaliser les contextes, décrit dans la section suivante.

10.1.2 Langage de description de la DÉMARCHE d'une méthode

Il a été défini trois types de contextes de **DÉMARCHE** : ceux représentés par une séquence (les *plans*), ceux représentant un ensemble de choix (les *choix*) et ceux étant des actions non décomposables (les *exécutables*). Le formalisme de description textuelle de ces différents types défini dans le projet NATURE a été décrit de façon abrégée dans [Plihon96] de la façon suivante.

<*X*> = • <*Y*> • <W> • ... • <*Z*>

Cette écriture permet de décrire de façon plus formelle la séquence d'un contexte <X> de type plan. Cette séquence est composée de l'ensemble des contextes (<Y>,<W>,…,<Z>). Prenons par exemple l'écriture suivante :

< Description du Problème ; Construire une Classe Objet > =

- *< Description du Problème ; Identifier une Classe Objet >*
- *< Classe Objet ; Décrire la Classe Objet >*.

Ce formalisme spécifie donc que le processus de construction d'une *Classe Objet* se fait en deux étapes qui sont l'identification et la description de cette classe.

<*X*> = " <*Y*> " <*W*> "... " <*Z*>

Ce formalisme permet de représenter de manière textuelle l'ensemble des choix proposés pour le contexte <X> de type choix. Ces choix sont présentés dans l'ensemble de contextes <Y> " <W> "... " <Z>). Pour la définition :

< Evénement ; Décrire l'événement > =

"< Evénement ; Décrire l'événement interne >

" < Evénement ; Décrire l'événement externe >

" < Evénement ; Décrire l'événement temporel >,

cela signifie qu'il existe trois possibilités lors de la description du concept d'événement qui sont les descriptions des événements internes, externes et temporels.

De plus, certains processus types peuvent être définis, comme la branche de l'arbre correspondant au processus de description d'un élément spécifique ou encore la racine de l'arbre. Les termes représentants ces processus sont décrits de la manière suivante.

$Proc_{[X]}$

Ce terme désigne le processus de description de l'élément X présent dans l'arbre de processus de la **DÉMARCHE** de la méthode.

Si nous définissons X comme le concept de *Classe Objet* de la méthode O*, *$Proc_{[Classe\ Objet]}$* représentera le processus <*Classe Objet, Décrire Classe Objet*>.

Racine ($Proc_{[M]}$)

Cette expression permet de désigner la racine de l'arbre de processus correspondant à la **DÉMARCHE** de construction du schéma objet.

Par exemple, la racine de l'arbre de processus de la méthode O* s'écrira *Racine ($Proc_{[O*]}$)* et correspondra au contexte <*Description du problème ; Construire le Schéma objet O**>.

Extension ($Proc_{[M]}$)

Ce formalisme ne s'utilise que dans le cas particulier où l'arbre de processus de la méthode que l'on souhaite étendre possède un contexte permettant de construire le schéma objet puis de l'étendre, ceci de façon séquentielle. Dans ce cas, cette expression correspondra au contexte représentant l'extension.

Prenons l'exemple de la méthode O* qui aurait déjà été étendue auparavant et qui possèderait donc déjà ce contexte. Le terme *Extension ($Proc_{[O*]}$)* représente alors le processus <*Schéma Objet, Etendre le schéma Objet*>.

Type-Parent-de ($Proc_{[X]}$)

Cette fonction permet de connaître le type du nœud père de la démarche de description de l'élément X. Par exemple, *Type-Parent-de (<Classe Objet ; Décrire Propriété>)* retourne le type PLAN puisque *Parent-de (<Classe Objet ; Décrire Propriété>)* est un nœud représentant la séquence de description d'une Classe Objet composé des séquences <*Classe Objet ; Décrire Propriété*>, <*Classe Objet ; Décrire Opération*> et <*Classe Objet ; Décrire Evénement Interne*>. Au contraire de *Type-Parent-de (<Evénement Interne; Décrire Evénement Interne>)* qui retourne le type CHOIX puisque *Parent-de (<Evénement Interne; Décrire Evénement Interne>)* est un nœud permettant de choisir une possibilité parmi les suivantes : <*Evénement Interne; Décrire Evénement Interne*>, <*Evénement Interne; Décrire Evénement Externe*>, <*Evénement Interne; Décrire Evénement Temporel*>.

10.1.3 Exemples de description de la situation et de la cible d'un patron d'extension de DÉMARCHE

Prenons par exemple une situation de méthode dont la **DÉMARCHE** serait représentée avec le formalisme graphique de NATURE par le schéma suivant. Cet exemple exprime le fait que la **DÉMARCHE** de description du concept de *Classe Calendrier* passe par l'identification et la description de ses composants : les concepts de *Propriété* et d'*Evénement temporel*.

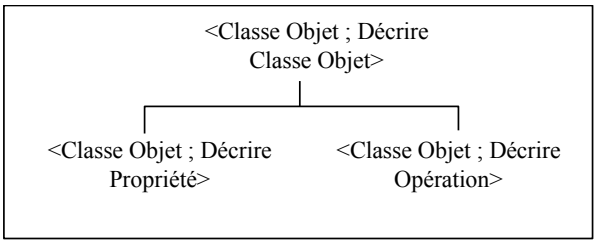


Figure 52: Exemple de situation de la partie DÉMARCHE avec le formalisme de NATURE

La formalisation de ce schéma au moyen du langage de description de la **DÉMARCHE** est la suivante.

```
<Classe Objet ; Décrire Classe Objet> =
    • <Classe Objet; Décrire Propriété>
    • <Classe Objet ; Décrire Opération>
```

Figure 53: Exemple correspondant avec le formalisme du langage de description de la DÉMARCHE

Appliquons maintenant sur cette situation un patron d'extension permettant d'intégrer dans la **DÉMARCHE** la construction du concept de *Contrainte*, composant de *Classe Objet*. La cible obtenue après application de ce patron d'extension est illustrée dans la figure suivante.

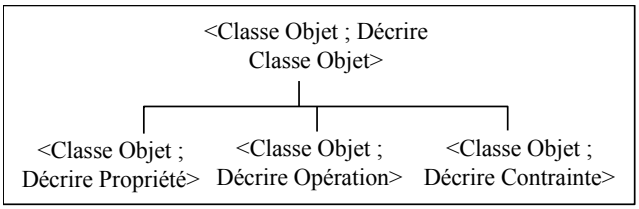


Figure 54: Exemple de cible de la partie DÉMARCHE avec le formalisme de NATURE

La description textuelle attachée à cette représentation est la suivante.

```
<Classe Objet ; Décrire Classe Objet> =
    • <Classe Objet; Décrire Propriété>
    • <Classe Objet ; Décrire Opération>
    • <Classe Objet ; Décrire Contrainte>
```

Figure 55: Exemple correspondant avec le formalisme du langage de description de la DÉMARCHE

10.2 Structure des intentions des patrons d'extension de DÉMARCHE

L'intention permet de définir ce que l'ingénieur de méthodes veut obtenir après exécution du patron d'extension. De la même manière que pour les descriptions de **PRODUIT**, un formalisme spécifique permet de décrire les intentions souhaitées.

10.2.1 Structure des intentions des patrons d'extension de la DÉMARCHE

Le formalisme utilisé pour décrire une intention a été défini à la section 8.1.1.2. Cette section exprime le fait qu'une intention est composée d'un verbe et de plusieurs paramètres pouvant représenter la cible, le résultat ou encore la manière à utiliser pour atteindre cette cible.

Il est à noter que la structure des intentions est différente selon le type du patron d'extension, c.-à-d. patron d'extension de **PRODUIT** ou de **DÉMARCHE**. Nous pouvons voir à la Figure 47 que l'intention d'un patron d'extension de **DÉMARCHE** a été définie de la façon suivante.

(Etendre)$_{Verb}$, par (Manière)$_{Man}$, (Proc$_{[M]}$)$_{obj}$ pour y intégrer (la construction de l'élément inséré dans le Produit)$_{res}$

Figure 56: Structure de l'intention d'un patron d'extension de la DÉMARCHE

On peut constater sur cette figure que le *verbe* (verb) correspondant aux patrons d'extension de la **DÉMARCHE**, comme d'ailleurs pour les patrons d'extension du **PRODUIT**, d'une méthode est toujours le verbe « Etendre ». La manière (Man) correspond à la façon dont on va modifier la méthode. L'*objet* (obj) concerné par l'extension est variable en fonction de la méthode d'origine et représente tout ou partie de sa **DÉMARCHE**. Le *résultat* (res) correspond à l'insertion de la **DÉMARCHE** de construction du nouvel élément (celui qui a été intégré lors de l'extension de la partie **PRODUIT**) dans la méthode.

10.2.2 Exemple de la structure de l'intention d'un patron d'extension de DÉMARCHE

Prenons l'exemple de l'intention suivante.

(Etendre)$_{Verb}$, par (COMPOSITION INTEGREE)$_{Man}$, (la construction des Classes Objets)$_{Obj}$ pour y intégrer (la construction des Contraintes)$_{Res}$

Figure 57: Exemple de décomposition d'intention

On peut déterminer dans cet exemple plusieurs points importants :

Ce que l'ingénieur de méthodes veut faire (le verbe), c'est *étendre* la méthode d'origine.

La façon dont il veut s'y prendre (la manière), c'est par *Intégration* du processus de construction d'un concept (ajouté dans la partie **PRODUIT** par *Composition*) dans la partie **DÉMARCHE** de la méthode.

Ce que le patron va modifier (l'Objet), c'est la partie de la **DÉMARCHE** de la méthode correspondant à la situation : *la construction du concept de Classe Objet.*

Ce que l'on intègre à cet objet (le résultat), c'est *la construction des Classes Temporelles*.

10.3 Description des opérateurs de transformation de la Démarche

Nous avons défini l'étape d'extension d'un concept de manière textuelle, il existe donc un langage de manipulation du **PRODUIT** et un langage de manipulation de la **DÉMARCHE** qui permettent la description du corps du patron d'extension.

Le formalisme de NATURE représente donc le modèle de processus comme un arbre de contextes. Chaque nœud de cet arbre représente un contexte qui est de type choix, de type plan ou de type exécutable. Il y a deux types d'arc : les arcs-plan (entre un contexte plan et l'un de ses contextes composants) et les arcs-choix (entre un contexte choix et l'une de ses possibilités).

10.3.1 Langage d'extension de la DÉMARCHE d'une méthode

Insérer nœud *NouveauSommet*

Insérer un nouveau contexte au processus se traduit par la greffe d'un nouveau sommet dans l'arbre. Par exemple, l'opérateur ***Insérer nœud*** *<Description du Problème **;** Définir contrainte d'unicité>* permettra d'ajouter le contexte de définition de contrainte d'unicité dans l'arbre. Il est cependant à noter que cet opérateur, même s'il insère un nœud, ne rattache pas celui-ci à l'arbre, et qu'il faudra donc utiliser un opérateur d'insertion de lien (*Insérer arc-plan* ou *Insérer arc-choix*) pour cela.

Renommer *AncienSommet* ***avec*** *NouveauSommet*

De la même façon que pour le **PRODUIT**, certaines parties du processus peuvent avoir besoin d'être renommée dans le but de spécifier une spécialisation du **PRODUIT**. Si l'on prend de nouveau l'exemple de l'extension temporelle des domaines, l'opérateur ***Renommer*** *< Description du Problème, Définir domaine>* ***avec*** *< Description du Problème, Définir domaine non temporel>* permettra de déplacer la branche de définition du domaine pour la spécialiser en tant que définition de domaine non temporel.

Supprimer *AncienSommet*

Il est parfois nécessaire de supprimer un contexte de la **DÉMARCHE** à suivre. Cela se traduit par une élimination d'un sommet de l'arbre. Par exemple, l'opérateur ***Effacer*** *< Description du Problème, Définir événement>* permettra de supprimer le contexte de définition d'un événement. Il est à noter que la suppression d'un sommet a pour conséquence la suppression de tous les liens qu'il pouvait avoir avec l'arbre.

Insérer arc-choix entre *SommetPère* ***et*** *SommetFils* ***avec argument*** *Argument*

L'extension du processus d'une méthode peut également passer par une modification des chemins possibles de l'arbre (si l'on a ajouté un nouveau sommet ou que l'on désire transformer le type d'un contexte). Ce formalisme particulier permet de rajouter une possibilité à un contexte choix, par ex. l'opérateur ***Insérer arc-choix entre*** *< Description du Problème, Définir type contrainte* ***et*** *< Description du Problème, Définir contrainte d'unicité>* ***avec argument*** *« l'ingénieur d'application veut contraindre l'unicité de l'objet »* permet de spécialiser le contexte de définition de contrainte avec le contexte de définition de contrainte d'unicité.

Insérer arc-plan entre *SommetPère* ***et*** *SommetFils*

De la même manière, il est possible de greffer un composant à la séquence d'un contexte plan. On peut illustrer ce formalisme avec l'exemple de l'opérateur suivant : ***Insérer arc-plan entre*** *<Contrainte, Décrire contrainte>* ***et*** *<Contrainte, Définir classification contrainte>* qui permet d'ajouter le contexte de définition de la classification d'une contrainte au concept de description de celle-ci.

Supprimer arc entre *SommetPère* ***et*** *SommetFils*

Au contraire des formalismes précédents qui permettent d'insérer un lien entre les contextes d'un arbre, celui-ci permet d'en supprimer un, c.-à-d. supprimer une alternative ou une composante d'un contexte. Par exemple, l'opérateur ***Supprimer arc entre*** *<Contrainte, Définir type contrainte>* ***et*** *<Pbstats, Définir contrainte d'unicité>* permet d'éliminer le lien se trouvant entre le contexte de définition de la contrainte d'unicité et de son contexte père.

Insérer branche *RacineBranche*

L'extension peut également rendre nécessaire le fait de greffer une branche entière de processus à l'arbre d'origine, c.-à-d. insérer un contexte et tous ses fils. Par exemple, l'opérateur ***Insérer branche*** *<Contrainte, Définir classification contrainte>* ajoutera à l'arbre toute la branche permettant de définir la classification d'une contrainte. Il est cependant à noter que, de la même façon que pour le formalisme d'insertion de sommet, celui-ci greffe une branche mais ne la relie pas encore à l'arbre. Pour effectuer cette dernière opération, il faudra exécuter l'un des opérateurs d'insertion de lien (*Insérer arc-choix* ou *insérer arc-plan*).

Supprimer branche *RacineBranche*

De la même manière, l'extension peut induire une suppression d'une branche entière de l'arbre. Par exemple, l'opérateur ***Supprimer branche*** *<Evénement, Décrire événement>* supprimera le contexte de description de l'événement, le lien le rattachant à son contexte père ainsi que tous ses fils se trouvant dans l'arbre.

Insérer Graphe-Précédence pour *Sommet*

Insérer un contexte plan dans l'arbre de processus entraîne le fait qu'il faut créer un graphe de précédence pour ce contexte, c.-à-d. spécifier l'ordonnancement de ses composantes. Par exemple, l'opérateur ***Insérer Graphe-Précédence pour*** *<Classe, Décrire Classe>* définira la séquence à suivre entre les contextes fils du contexte de description d'une classe.

***Changer Graphe-Précédence pour** Sommet*

De la même façon, le fait de modifier les fils d'un contexte plan entraîne la modification de son graphe de précédence, c.-à-d. modifier l'ordonnancement de ses composantes. Par exemple, l'opérateur ***Changer Graphe-Précédence pour*** <*Contrainte, Décrire contrainte*> permettra de modifier le graphe de précédence associé au contexte de description d'une contrainte.

10.3.2 Exemple de description des opérateurs de transformation d'une DÉMARCHE

Prenons l'exemple d'une partie de la **DÉMARCHE** d'une méthode permettant de décrire une *Classe Objet* en identifiant et en décrivant sa composition avec les deux concepts de *Propriétés* et d'*Opérations*. La figure suivante décrit cette situation par le biais du langage de description de la **DÉMARCHE**. Cet exemple est illustré dans la Figure 58.

```
<Classe d'Objet ; Décrire Classe d'Objet> =
     • <Classe d'Objet ; Décrire Propriétés>
     • <Classe d'Objet ; Décrire Opérations>)
```

Figure 58: Description d'une partie de la DÉMARCHE

Modifions maintenant cette **DÉMARCHE** pour introduire un nouveau nœud dans la séquence de description d'une Classe d'Objet : l'identification et la description du concept de *Contrainte*. La figure suivante illustre les opérateurs à effectuer sur cette **DÉMARCHE** pour réaliser cette modification avec le langage d'extension.

```
Insérer arc-plan entre <Classe d'Objet ; Décrire Classe d'Objet> et <Classe
d'Objet ; Identifier et Décrire Contraintes>
Changer Graphe-Précédence pour <Classe d'Objet ; Décrire Classe d'Objet>
```

Figure 59: Modification de la DÉMARCHE

A la sortie du patron d'extension, la **DÉMARCHE** a obtenu la nouvelle configuration illustrée à la Figure 60.

```
<Classe d'Objet ; Décrire Classe d'Objet> =
      • <Classe d'Objet ; Identifier et Décrire Propriétés>
      • <Classe d'Objet ; Identifier et Décrire Opérations>
      • <Classe d'Objet ; Identifier et Décrire Contraintes>
```

Figure 60: Description de la partie de la DÉMARCHE modifiée

10.4 Exemple de l'interface d'un patron d'extension de DÉMARCHE

L'interface du patron d'extension utilisé dans cette section peut donc être visualisée comme dans la figure suivante.

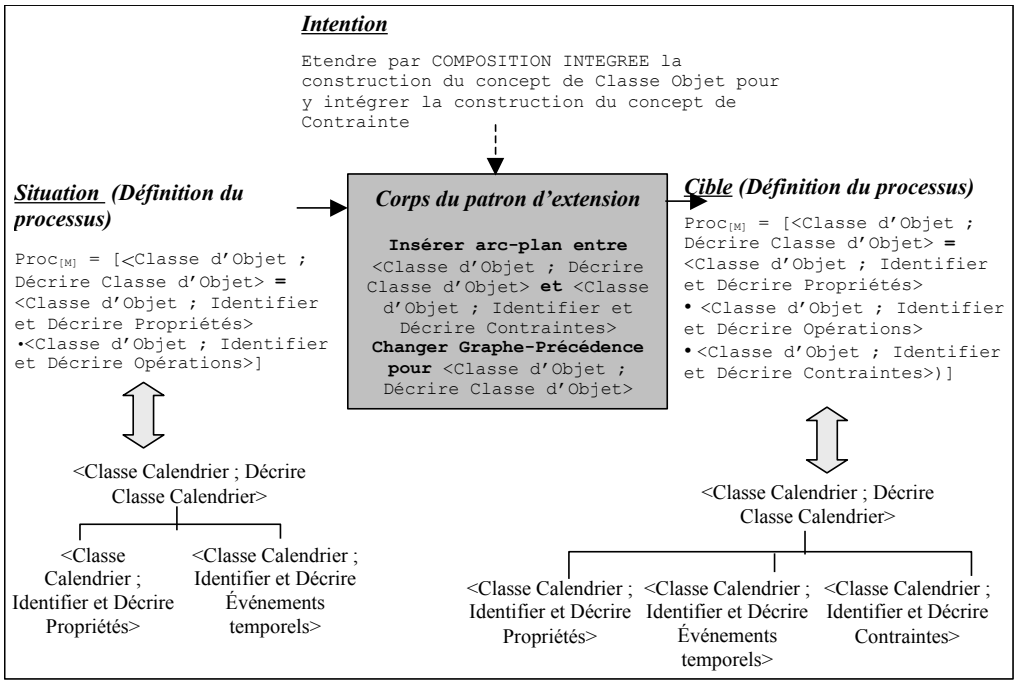


Figure 61: Exemple d'exécution du corps d'un patron d'extension

10.5 Exemple d'application d'une extension de la DÉMARCHE d'une méthode

10.5.1 Problème

Prenons la suite de l'exemple décrit dans la section 9.5. L'ingénieur de méthodes utilise la méthode O*, dont il a étendu la partie **PRODUIT** pour y intégrer le concept de *Classe Calendrier* (par Remplacement), pour avoir la possibilité de définir plusieurs référentiels temporels dans son application. Si l'ingénieur veut également étendre la partie **DÉMARCHE** de la méthode O*, il se doit d'appliquer également un autre patron d'extension.

10.5.2 , rbre de processus de la DÉMARCHE de la méthode d'origine

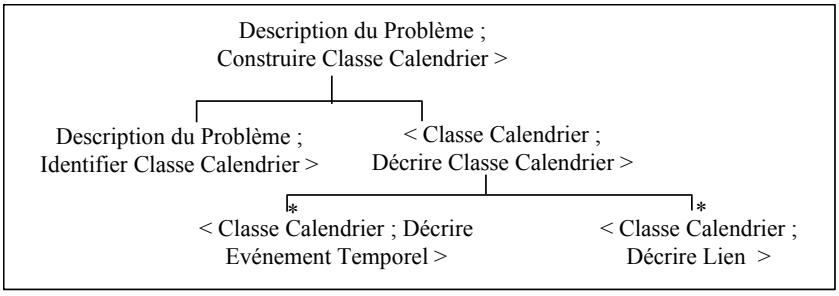


Figure 62: Partie de l'arbre de processus de la méthode O*

La **DÉMARCHE** de construction du concept de *Classe Calendrier* dans l'arbre de processus de la méthode O* est représentée par un plan permettant d'exécuter deux étapes : la description des

Evénements Temporels liés au Calendrier et la description des liens que ce concept a avec le reste du modèle.

L'ingénieur de méthodes a étendu la partie **PRODUIT** de la méthode avec le concept de Classe Calendrier par Remplacement. Il choisit ensuite d'étendre la **DÉMARCHE** de la méthode en intégrant directement les changements dans l'arbre de processus.

10.5.3 Description du patron d'extension

L'interface du patron d'extension d'une méthode au référentiel temporel est illustrée dans la Figure 63 suivante.

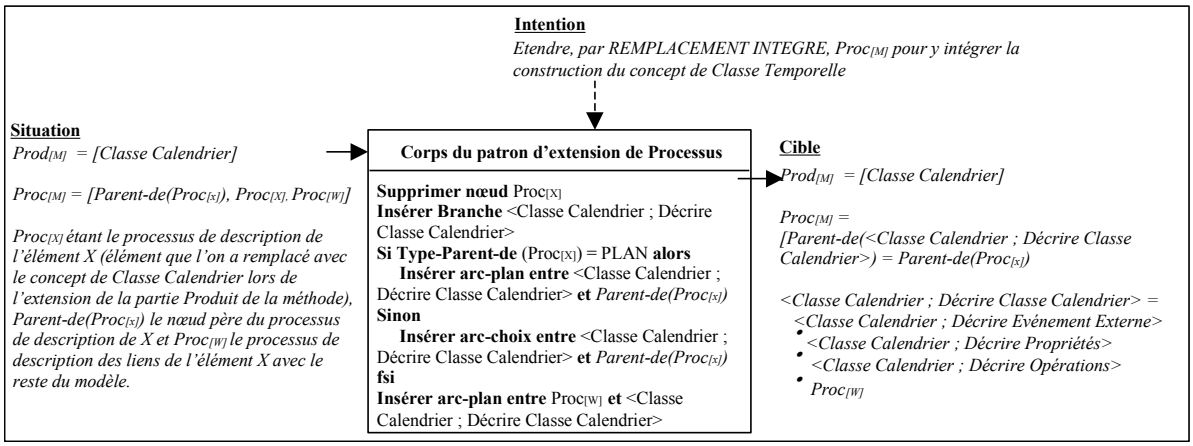


Figure 63: Interface du patron d'extension d'une méthode au référentiel temporel

Signature formelle

On peut constater sur cette figure que l'intention du patron d'extension est bien d'étendre la méthode d'origine pour prendre en compte, de façon intégrée dans l'arbre de processus, la construction du concept de *Classe Calendrier*. La situation du patron est donc représentée par la partie **PRODUIT** de la méthode ayant été modifiée ainsi que par la partie **DÉMARCHE** non encore modifiée. La cible représente ici la méthode prenant en compte la modification, tant au niveau **PRODUIT** qu'au niveau **DÉMARCHE**.

Corps

Les opérateurs de modification à effectuer sur la méthode d'origine seront donc les suivants : ***Supprimer nœud*** *Proc[X] ;*
Insérer Branche *<Classe Calendrier ; Décrire Classe Calendrier> ;*
Si *Type-Parent-de (Proc[X]) = PLAN* ***alors***
Insérer arc-plan entre *<Classe Calendrier ; Décrire Classe Calendrier>* ***et*** *Parent-de(Proc[x])*
Sinon
Insérer arc-choix entre *<Classe Calendrier ; Décrire Classe Calendrier>* ***et*** *Parent-de(Proc[x])*
fsi

Insérer arc-plan entre *<Classe Calendrier ; Décrire Classe Calendrier>* ***et*** *Parent-de($Proc_{[x]}$) ;* ***Insérer arc-plan entre*** *$Proc_{[W]}$* ***et*** *<Classe Calendrier ; Décrire Classe Classe Calendrier>*.

Signature informelle

Le domaine d'application de ce patron est celui des applications temporelles. Les heuristiques qui lui sont associées sont les suivantes.

> La partie **PRODUIT** de la méthode d'origine a été étendue pour y ajouter le concept de Classe Calendrier par remplacement d'un concept existant. Pour étendre la partie **DÉMARCHE** de la méthode, il est nécessaire de remplacer également le processus de construction de l'ancien concept par le processus de construction du nouveau concept de Classe Calendrier. Pour cela, il suffit de supprimer le nœud correspondant et de greffer la nouvelle branche à sa place (rattachement au même nœud père et conservant le même nœud fils pour la construction des liens rattachant le nouveau concept au modèle).

Les forces et les faiblesses de ce patron sont décrites de la manière suivante.

> Le point fort de l'application de ce patron est de permettre à l'ingénieur de méthodes, non seulement la manipulation de plusieurs référentiels temporels grâce à l'extension de la partie **PRODUIT** , mais également une manipulation plus aisée de la construction de ceux-ci. En effet, l'arbre de processus correspondant à la démarche de construction du schéma sera plus cohérent avec cette modification et pourra donc être utilisé de façon plus détaillée.

10.5.4 Instanciation du patron d'extension pour la méthode O*

L'interface de cette extension est illustrée à la figure suivante.

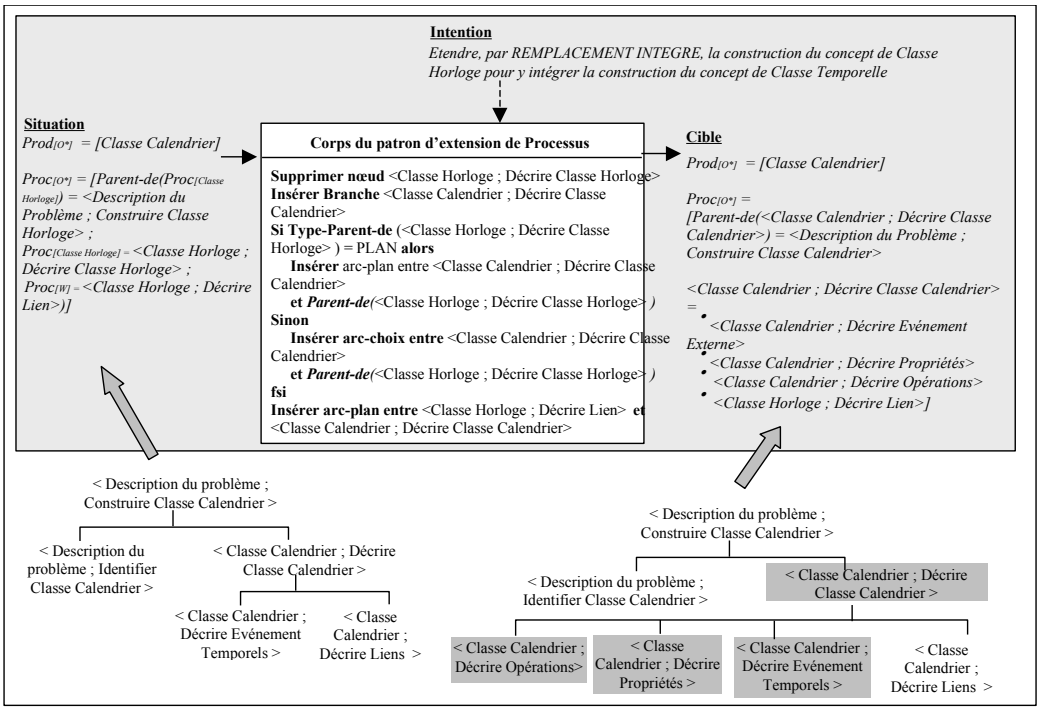


Figure 64: Application du patron d'extension de la méthode O* permettant d'intégrer la construction du concept de Classe Calendrier dans la DÉMARCHE.

L'interface du patron de cet exemple définit les aspects spécifiques du patron.

La situation du **PRODUIT** et du processus de la méthode O* lors de l'application du patron ($Prod_{[O*]}$, $Proc_{[O*]}$) est donc la suivante :

```
Prod[O*] = [Classe Calendrier]

Proc[O*] = [Parent-de(Proc[Classe Horloge]) = <Description du Problème ; Construire
Classe Calendrier> ;
          Proc[Classe Horloge] = <Classe Horloge ; Décrire Classe Horloge> ;
          Proc[W] = <Classe Horloge ; Décrire Lien>)]
```

L'intention du patron d'extension est :

```
Etendre, par REMPLACEMENT INTEGRE, la construction du concept de Classe
Horloge pour y intégrer la construction du concept de Classe Calendrier
```

La situation du **PRODUIT** obtenue après extension ($Prod_{[O* etendu]}$)est :

```
Prod[O*] = [Classe Calendrier]

Proc[O*] = [Parent-de(<Classe Calendrier ; Décrire Classe Calendrier>) =
<Description du Problème ; Construire Classe Calendrier>

<Classe Calendrier ; Décrire Classe Calendrier> =
   • <Classe Calendrier ; Décrire Evénement Externe>
```

- <Classe Calendrier ; Décrire Propriétés>
- <Classe Calendrier ; Décrire Opérations>
- <Classe Horloge ; Décrire Lien>]

Le corps du patron est composé de l'ensemble des opérateurs suivants.

```
Supprimer nœud <Classe Horloge ; Décrire Classe Horloge>
Insérer Branche <Classe Calendrier ; Décrire Classe Calendrier>
Si Type-Parent-de (<Classe Horloge ; Décrire Classe Horloge> ) = PLAN alors
     Insérer arc-plan entre <Classe Calendrier ; Décrire Classe Calendrier>
     et Parent-de(<Classe Horloge ; Décrire Classe Horloge> )
Sinon
     Insérer arc-choix entre <Classe Calendrier ; Décrire Classe Calendrier>
     et Parent-de(<Classe Horloge ; Décrire Classe Horloge> )
fsi
Insérer arc-plan entre <Classe Calendrier ; Décrire Classe Calendrier> et
<Description du Problème ; Construire Classe Calendrier>
Insérer arc-plan entre <Classe Horloge ; Décrire Lien> et <Classe
Calendrier ; Décrire Classe Calendrier>
```

10.5.5 Arbre de processus de la méthode étendue

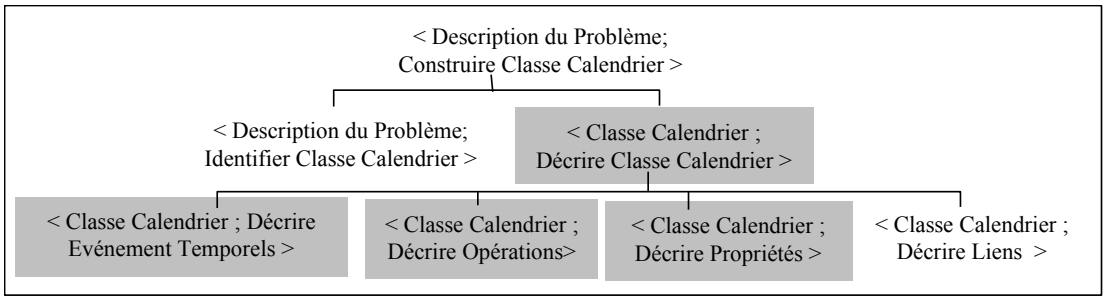


Figure 65: Partie de l'arbre de processus de la méthode O* étendue

L'application du patron d'extension de la **DÉMARCHE** de la méthode O* a donc fourni le résultat escompté par l'ingénieur de méthodes et lui a permis d'étendre cette méthode pour rendre sa **DÉMARCHE** adaptée à son nouveau **PRODUIT**.

Chapitre V : Organisation des patrons d'extension

Introduction

Le chapitre précédent définit la spécification et la description des patrons d'extension utilisés dans ce mémoire, cette section se focalise maintenant sur leur organisation spécifique dans une optique de réutilisation.

En effet, les patrons d'extension définis par l'ingénieur de méthodes sont stockés dans une bibliothèque à partir de laquelle ils pourront être réutilisés mais certains patrons possèdent entre eux une relation de précédence qui permet de conserver une bonne cohérence de la méthode générée. Il est donc nécessaire de les organiser de façon adéquate dans la bibliothèque pour permettre de les exécuter dans un ordre correct. Cet ordre peut être obtenu en utilisant la technique de représentation des patrons par une *carte d'extension* [Rolland99][Benjamen99].

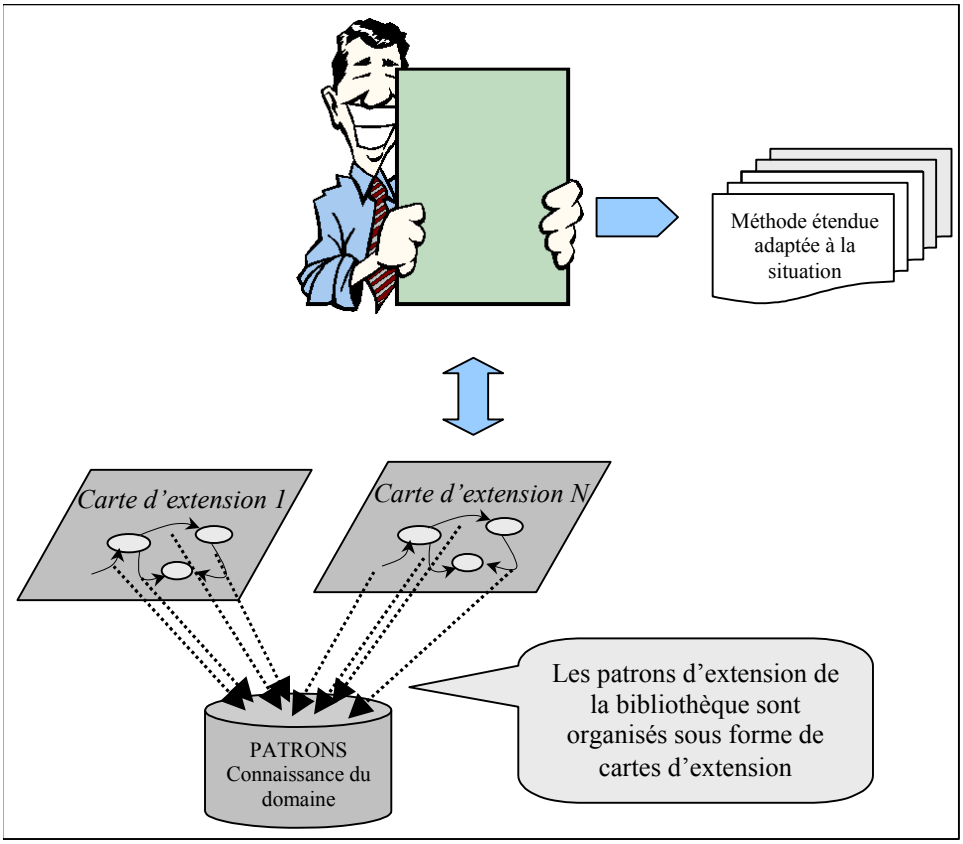


Figure 66: Processus d'organisation des patrons

Comme l'indique la figure précédente, l'ingénieur de méthode utilise une carte d'extension pour étendre la méthode d'origine dans le but d'exécuter les patrons d'extension lui permettant d'obtenir une méthode adaptée à ses besoins mais ayant également conservé une bonne cohérence. Ce chapitre décrit la spécification des cartes d'extension, ainsi que la manière de les utiliser et de les construire à partir des patrons d'extension de la bibliothèque.

11 Carte d'extension

La Figure 67 illustre l'application de la technique d'organisation, par le biais d'une carte d'extension, sur plusieurs patrons d'extension de méthodes aux applications temporelles.

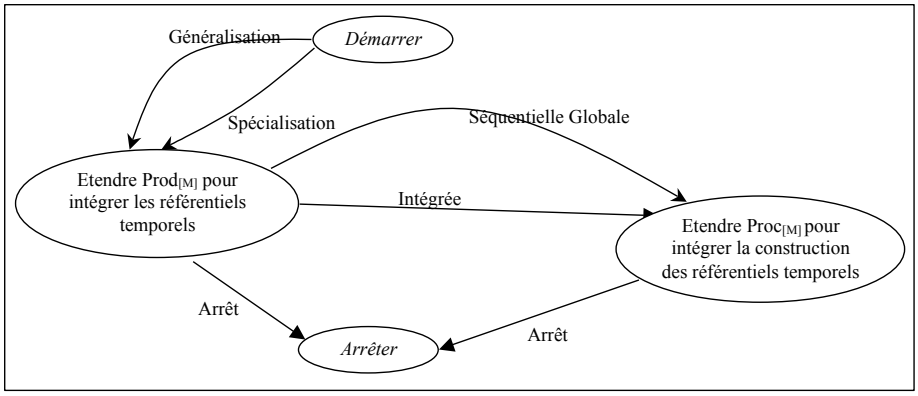


Figure 67: Exemple d'une partie de carte d'extension

On peut voir sur cet exemple qu'une carte d'extension est représentée par un graphe composé de nœuds et d'arcs. Les nœuds représentent les patrons d'extension que l'on veut organiser dans la bibliothèque. Ce sont les arcs qui permettent de donner un certain ordre d'exécution de ces patrons . Un chemin d'exécution déterminé par une carte commence toujours par le nœud *Démarrer* et se termine toujours par le nœud *Arrêter*. Les arcs permettent de naviguer de patrons d'extension en patrons d'extension par l'application d'un certain nombre de stratégies (chacune étant représentée par un arc spécifique entre deux patrons). Un patron d'extension spécifique est donc représenté dans la carte par une section, i.e. un arc reliant deux nœuds, autrement dit par l'application d'une stratégie permettant de partir d'une intention source pour atteindre une intention cible.

L'ingénieur de méthodes effectue donc l'extension de sa méthode en naviguant dans la carte. L'exemple visualisé à la Figure 67 indique que deux stratégies d'extension sont possibles pour naviguer du nœud *Démarrer* au nœud *Etendre Prod[M] pour intégrer les référentiels temporels*. Ces deux stratégies sont la *Généralisation* et la *Spécialisation*. L'ingénieur de méthodes peut donc choisir celle qui lui convient le mieux par rapport à la méthode qu'il souhaite étendre. Par exemple, si sa méthode contient déjà un concept qui pourrait généraliser celui de référentiel temporel, alors la stratégie la mieux adaptée à l'extension sera la *Spécialisation*. Cependant, si la méthode contient, non pas un concept qui pourrait généraliser celui de référentiel temporel mais qui lui serait similaire, alors l'extension la plus adaptée serait de généraliser ce concept et de lui affecter celui de référentiel temporel comme nouvelle spécialisation (stratégie *Généralisation*). Une fois son choix de stratégie effectué, l'ingénieur de méthodes peut donc exécuter le patron d'extension correspondant.

Il se trouve donc maintenant au niveau du nœud *Etendre Prod[M] pour intégrer les référentiels temporels* de la carte d'extension. Il a maintenant deux choix de navigation possibles : soit il continue à étendre sa méthode en naviguant jusqu'au nœud *Etendre Proc[M] pour intégrer la construction des référentiels temporels*, soit il termine sa navigation en atteignant le nœud *Arrêter*. Dans le premier cas, il doit choisir l'une des deux stratégies lui permettant d'atteindre le nœud, soit par une extension

Intégrée, soit par une extension *Séquentielle Globale*. Dans le second cas, il lui suffit d'utiliser la stratégie *d'Arrêt* pour atteindre le nœud *Arrêter*.

La manière de naviguer dans une carte d'extension est la même quelle que soit la carte. L'ingénieur de méthodes part du nœud *Démarrer*, choisit une stratégie afin d'atteindre un autre nœud et ainsi de suite jusqu'à atteindre le nœud *Arrêter* qui lui permet de terminer l'exécution de son extension. Cette technique est un moyen d'organiser simplement les patrons d'extension sous forme de sections dans la carte d'extension de façon à les retrouver et à les utiliser de la meilleure façon possible.

La suite de ce chapitre définit dans un premier temps la spécification des cartes d'extension (section 12), dans un deuxième temps l'utilisation de celles-ci en tant que moyen de navigation (section 13) et finalement la technique de construction de ces cartes à partir des patrons d'extension (section 14).

12 Spécification d'une carte d'extension

12.1 Carte d'extension

La carte permet d'obtenir un ordonnancement non déterministe d'intentions et de stratégies. En effet, le processus d'ingénierie est orienté intention car, à tout moment, l'ingénieur de méthodes a des intentions, des objectifs qu'il veut réaliser. De plus, il y a en général plusieurs stratégies pour réaliser la même intention. La Figure 68 illustre la structure d'une carte d'extension en utilisant le formalisme d'UML.

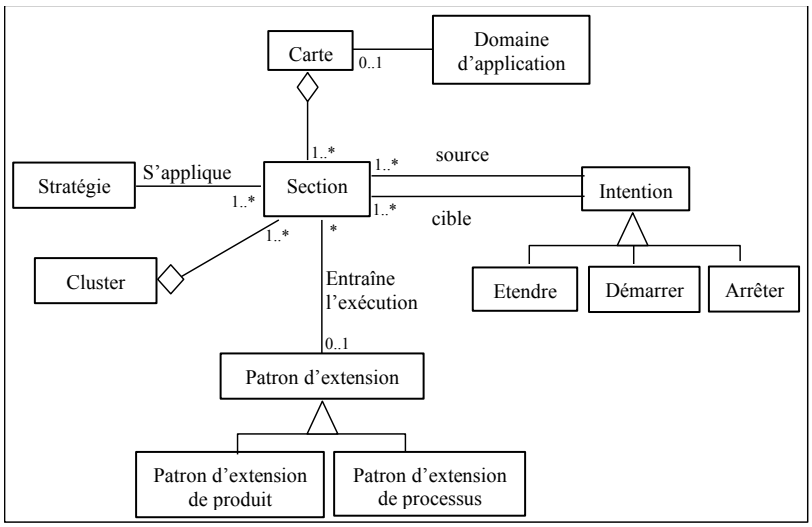


Figure 68: Structure d'une carte d'extension

Comme il a été défini au chapitre III, un patron est défini pour un domaine d'application spécifique. En effet, chaque patron correspond à la solution d'un problème posé pour un certain type d'application.

L'ingénieur de méthodes souhaite étendre sa méthode d'origine lorsqu'il se retrouve confronté à une situation où la méthode n'est pas adaptée au domaine d'application. En effet, selon le domaine dans

laquelle l'application est développée, les besoins de l'ingénieur seront différents. Les patrons d'extension sont donc regroupés dans la bibliothèque par rapport à leur domaine. Ils sont ensuite organisés entre eux grâce à une carte qui correspondra donc à un domaine particulier ; il y aura, par exemple, la carte d'extension aux applications temporelles, la carte d'extension aux applications multi-agents, etc.

12.2 Section

La carte comporte un ensemble de sections correspondant chacune à un triplet < Intention source, Intention cible, Stratégie >. Ces intentions cibles et sources sont définies de la même manière que la notion d'intention décrite au chapitre III. La carte contient deux intentions particulières appelées *Démarrer* et *Arrêter*. Lorsque l'intention cible d'une section est *Etendre*, cela signifie qu'elle entraînera l'exécution d'un patron d'extension pouvant concerner une extension de la partie **PRODUIT** ou une extension de la partie **DÉMARCHE** de la méthode. En effet, une extension d'une méthode avec un nouveau concept devrait, pour conserver sa complétude, étendre les deux parties de cette méthode (le modèle et la démarche).

Une carte se représente par un graphe étiqueté et dirigé. Les nœuds correspondent aux intentions et les arcs aux stratégies. Cette représentation graphique visualise le fait que le guidage du processus d'ingénierie consiste à naviguer dans la carte depuis l'intention *Démarrer* jusqu'à l'intention *Arrêter* (le nœud *Démarrer* permet de commencer la navigation dans la carte alors que le nœud *Arrêter* permet de terminer cette navigation). La Figure 69 visualise une partie d'une carte d'extension. Dans cet exemple particulier correspondant à une section spécifique de la carte, on peut voir que l'intention de départ est d'*Etendre* $Prod_{[M]}$ *pour intégrer les référentiels temporels* (le formalisme $Prod_{[M]}$ correspond à la partie Produit de la méthode M), que l'intention d'arrivée est d'*Etendre* $Prod_{[M]}$ *pour intégrer les domaines temporels* et que la stratégie à appliquer sur la situation de départ pour atteindre la cible est la stratégie d'*Extension par Composition*.

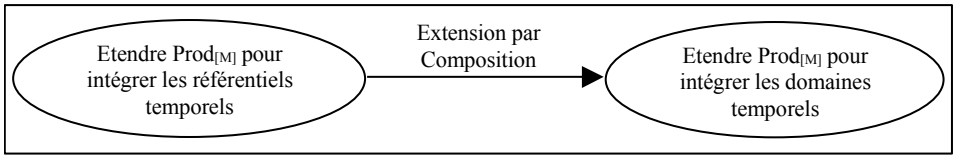


Figure 69: Section d'une carte d'extension

La manière spécifique d'accomplir une intention est capturée dans une section de la carte alors que les diverses sections qui ont les mêmes intentions de départ et d'arrivée définissent les différentes stratégies pouvant être adoptées pour accomplir l'intention d'arrivée (ces sections peuvent être regroupées sous le concept de *Cluster*). De la même façon, il peut y avoir différentes sections ayant comme source une même intention et comme arrivées des intentions différentes. Celles-ci montrent les différentes intentions qui peuvent être atteintes après la réalisation de l'intention de départ (c.-à-d.. quels sont les patrons d'extension qui peuvent être exécutés après l'exécution du premier patron).

Une carte est une structure de navigation dans le sens où elle permet à l'ingénieur d'application de déterminer un chemin d'intentions en partant de l'intention *Démarrer* pour finalement aboutir à

l'intention *Arrêter*. La carte contient un nombre fini de chemins. Aucun enchaînement de sections inclus dans la carte n'est recommandé « a priori » mais l'approche suggère une construction dynamique du chemin réel en naviguant dans la carte. Cette construction est donc faite en fonction des situations rencontrées.

12.3 Cluster

Comme il a été dit plus haut, les sections de la carte peuvent être regroupées sous forme de Cluster pour simplifier l'écriture et la lecture des cartes d'extension [Rolland00]. Dans la Figure 70, les six sections possibles entre les deux intentions de cette partie de la carte peuvent être regroupées avec un cluster correspondant au parcours entre ces deux intentions par l'exécution d'une stratégie parmi les suivantes : Extension par Spécialisation, Extension par Généralisation, Extension par Composition, Extension par Décomposition, Extension par Référence ou Extension par Remplacement.

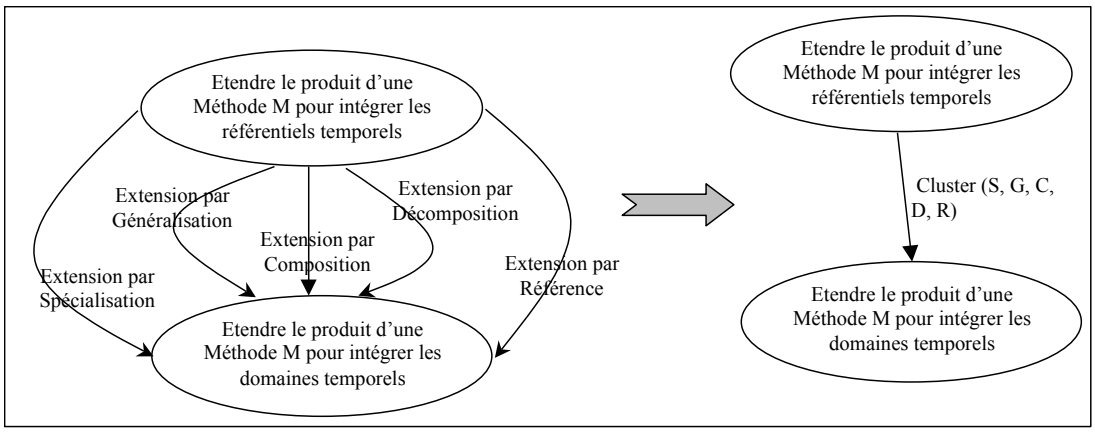


Figure 70: Exemple de Cluster

12.4 Patron

La notion de patron d'extension rattaché à chaque section d'une carte d'extension rejoint la notion d'Aide méthodologique décrite dans [Benjamen99] ainsi que celle de Directive de réalisation d'intention (ou *Intention Achievement Guideline*) de [Ralyté99]. La Figure 71 montre la correspondance existante entre la description d'une section et le patron qui lui est associé.

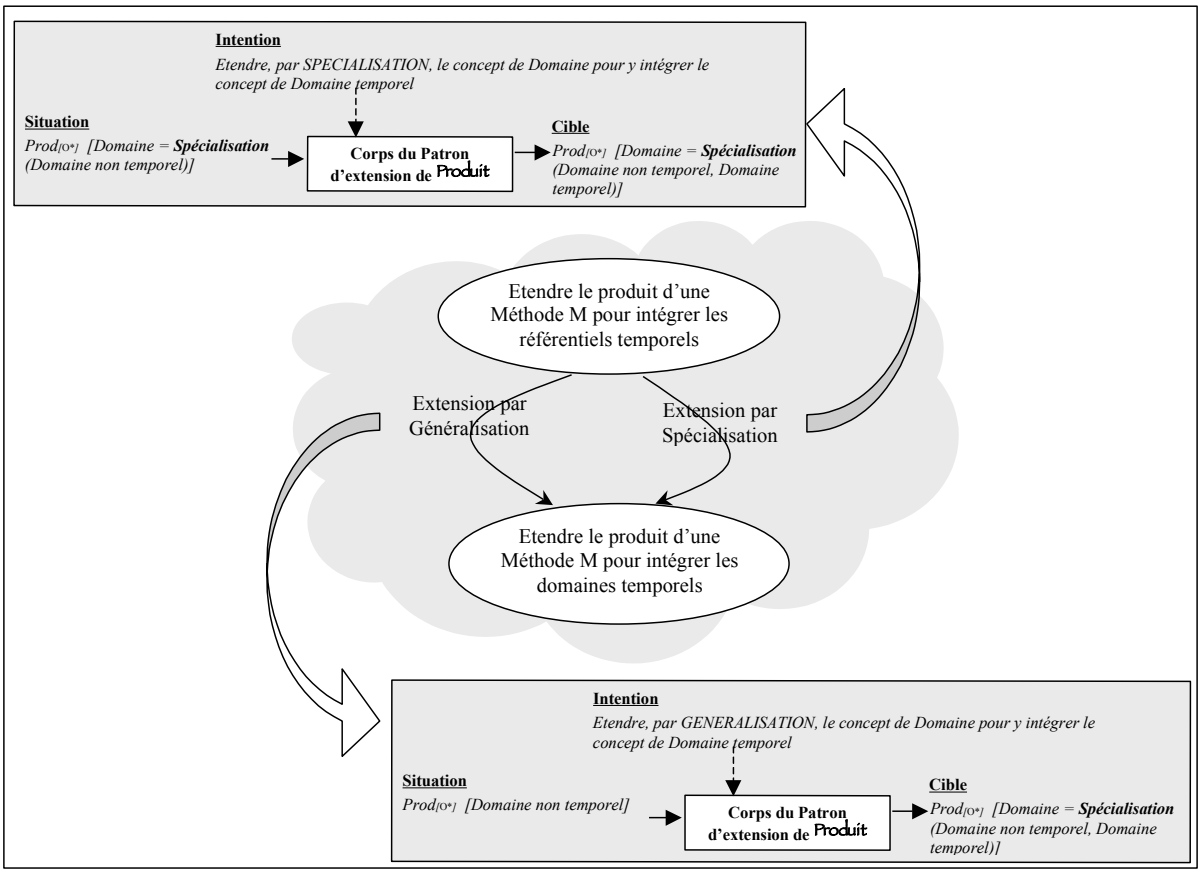


Figure 71: Exemple de Patrons organisés par une carte d'extension

Ces patrons, dont la description a été donnée dans le chapitre précédent, permettent l'exécution de l'extension sur la méthode d'origine. C'est à l'intérieur des patrons qu'est encapsulée toute la connaissance nécessaire à une extension cohérente d'une méthode.

13 Utilisation d'une carte d'extension

13.1 Règles d'utilisation

L'utilisation d'une carte d'extension peut se résumer en trois étapes spécifiques : le démarrage, la navigation dans la carte et l'arrêt de l'exécution.

Démarrage

L'ingénieur de méthodes commence l'exécution de la carte d'extension par une section dont l'intention source est l'intention *Démarrer*. Il choisit ensuite une intention cible parmi toutes les sections possibles (dont l'intention source est *Démarrer*). Selon la situation de la méthode d'origine, l'ingénieur choisit la stratégie d'extension la plus adaptée, ce qui lui permet donc d'exécuter le patron d'extension correspondant qui réalisera l'intention cible désirée.

Navigation dans la carte

Chaque exécution d'un patron permet à l'ingénieur de méthodes de réaliser une intention générique de la carte d'extension. Cette intention devient donc une intention source d'où il doit naviguer pour atteindre une autre intention (cette fois-ci une intention cible). De la même manière que pour la première étape, il doit tout d'abord choisir une intention cible parmi celles qui lui sont proposées par la carte. Ensuite, l'ingénieur de méthodes choisit une stratégie d'extension adaptée à la situation de la méthode d'origine. Une fois cette intention atteinte (par l'exécution du patron correspondant), la navigation reprend sur le même principe. Il est à noter qu'après toute extension effectuée, l'ingénieur de méthodes peut soit continuer de naviguer à partir de l'intention atteinte, soit reprendre la navigation à partir du nœud *Démarrer*.

Arrêt

Une fois que l'ingénieur de méthodes a atteint son objectif initial et qu'il souhaite arrêter l'exécution de la carte d'extension, il choisit l'intention cible *Arrêter* (atteignable par chacune des intentions génériques de la carte) qu'il réalise grâce à la stratégie intitulée *stratégie d'arrêt*.

13.2 Exemple d'utilisation d'une carte d'extension

L'exemple de la Figure 72 illustre une partie de la carte d'extension correspondant à l'application de cette approche au domaine des applications temporelles. Nous n'avons pris en compte ici que les patrons permettant d'intégrer les concepts de référentiel et de domaines temporels dans la partie **PRODUIT** et la construction de ces concepts dans la partie **DÉMARCHE**.

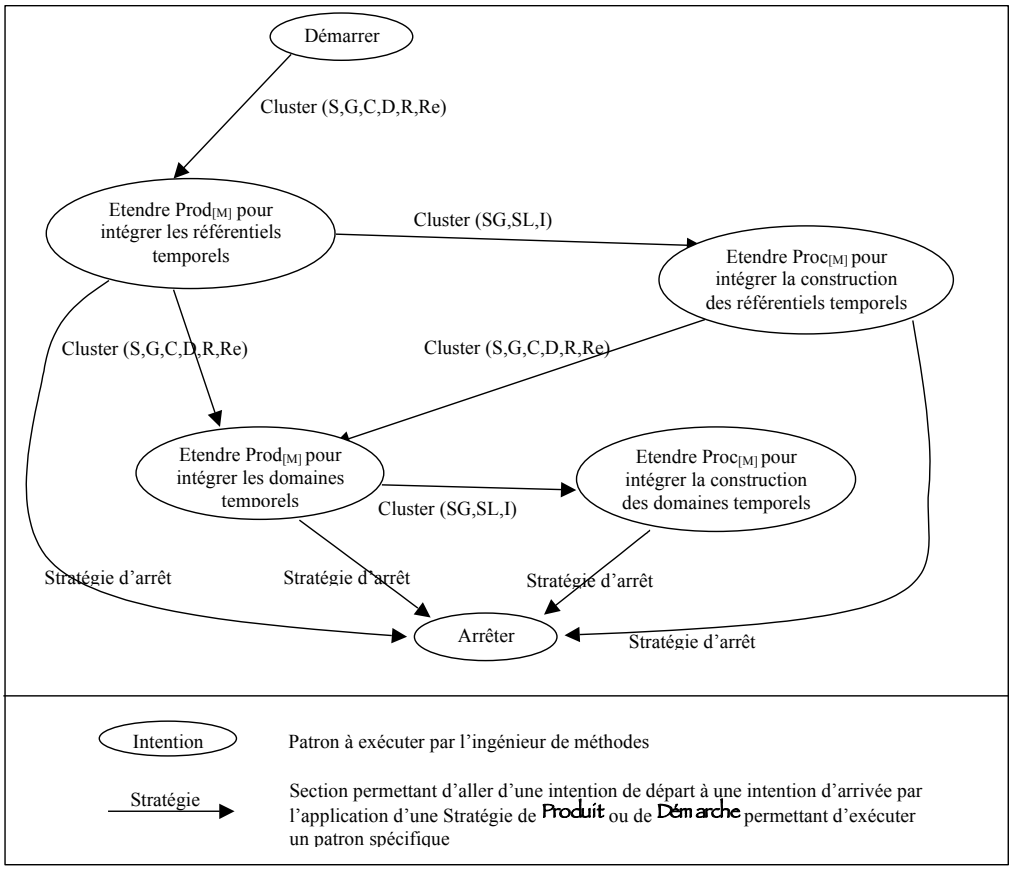


Figure 72: Partie de la carte d'extension aux applications temporelles

On peut remarquer qu'il existe six chemins possibles de navigation entre les intentions *Démarrer* et *Arrêter*. Certains chemins permettront de n'étendre que le **PRODUIT**, d'autres étendront l'ensemble de la méthode (**PRODUIT** et **DÉMARCHE**), certains n'intégreront que les concepts spécifiques aux référentiels temporels, d'autres encore intégreront également ceux des domaines temporels, etc. L'ingénieur de méthodes est laissé libre de choisir le chemin qui lui convient le mieux pour naviguer dans cette carte d'extension.

Cependant, on peut remarquer que certains enchaînements entre patrons ne sont pas autorisés par la carte. En effet, il n'est, par exemple, pas possible à l'ingénieur d'étendre la partie **DÉMARCHE** de sa méthode si il n'a pas étendu la partie **PRODUIT** auparavant. De la même manière, il ne lui est pas permis d'introduire les concepts de domaines temporels s'il n'a pas tout d'abord introduit ceux de référentiels temporels ; en effet, ces derniers sont nécessaires à la construction des premiers. L'utilisation de la carte d'extension induit donc une certaine cohérence de la méthode puisqu'elle guide l'ingénieur pas à pas dans l'exécution des patrons d'extension.

Prenons l'exemple d'un ingénieur de méthodes qui voudrait étendre sa méthode d'origine en lui intégrant le concept de domaine temporel. La carte d'extension ne lui permet pas d'exécuter directement le patron d'extension concernant ce point particulier, il va donc devoir intégrer le concept de référentiel temporel préalablement.

1ère étape : L'ingénieur de méthodes commence l'exécution de la carte avec l'intention *Démarrer*.

2ème étape : La seule intention cible qui lui est autorisée est celle permettant d'intégrer le concept de référentiel temporel. La situation de sa méthode d'origine lui permet de choisir sa stratégie d'extension parmi les suivantes : Spécialisation, Généralisation, Composition, Décomposition, Référence, Remplacement. Il choisit la stratégie d'extension par Spécialisation puisque sa méthode contient le concept de *Classe* spécialisée en *Classe Objet* et *Classe Acteur* auquel il ajoute donc le concept de *Classe Calendrier*.

3ème étape : L'ingénieur de méthodes a deux possibilités d'intention cible : *Etendre* $Proc_{[M]}$ *pour intégrer la construction des référentiels temporels* ou *Etendre* $Prod_{[M]}$ *pour intégrer les domaines temporels*. Il décide de n'étendre que le **PRODUIT** de sa méthode et se désintéresse de la partie **DÉMARCHE**, il choisit donc d'étendre directement le **PRODUIT** de sa méthode avec le concept de domaine temporel. Dans ce cas, la situation de sa méthode le contraint à choisir la stratégie d'extension par Généralisation puisque la méthode contient le concept de *Domaine non temporel* qu'il va donc généraliser en concept de *Domaine* pour lui ajouter une nouvelle spécialisation qui sera le concept de *Domaine temporel*.

4me étape : L'ingénieur de méthodes a atteint son objectif. Il décide de s'arrêter à la suite de l'exécution de ce patron et choisit donc la stratégie d'arrêt pour parvenir à l'intention *Arrêter*.

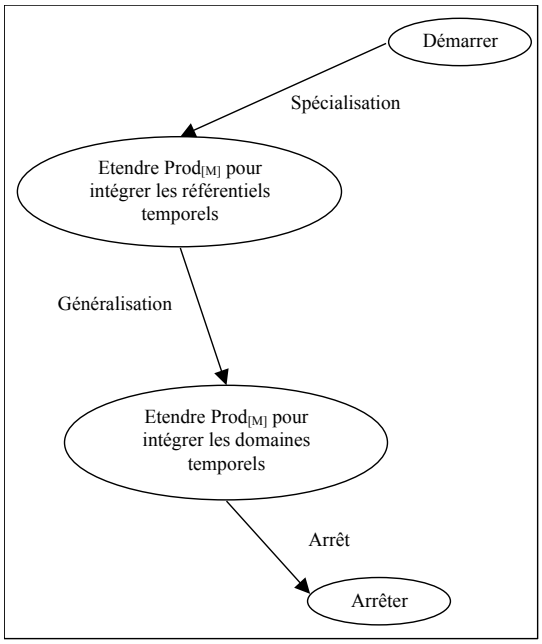


Figure 73: Chemin suivi dans la carte d'extension

14 Organisation des patrons par le biais des cartes d'extension

Un patron est décrit, ainsi qu'il a été expliqué dans le chapitre précédent, à l'aide de son interface, de sa signature informelle et de son corps. Les deux premières parties (chapitre III) sont composées des spécifications permettant la réutilisation de ce patron, notamment les détails concernant son *domaine d'application* et sa *signature*. Ce sont ces détails qui permettent de définir une carte d'extension pour un domaine d'application donné.

14.1 Règles d'organisation

Certaines étapes sont nécessaires pour réaliser une bonne carte d'extension. Ces étapes ont été découpées en un certain nombre de règles à suivre lors de cette définition.

Il est tout d'abord nécessaire de définir le domaine d'application qui nous intéresse pour pouvoir créer la carte d'extension associée. Les patrons d'extension sont classés dans la bibliothèque selon leur domaine d'application. Pour créer une carte d'extension, il faut commencer par extraire tous les patrons se référant à ce domaine spécifique.

Tous les patrons d'une carte doivent être rattachés au même domaine d'application.

Nous avons défini dans la section 12.2 que toute carte d'extension contenait au moins deux intentions prédéfinies, celle concernant le démarrage de la carte et celle permettant son arrêt.

Toute carte d'extension définie pour un domaine d'application contient au minimum les deux intentions *Démarrer* et *Arrêter*. Cependant, pour des raisons de clarté, l'intention *Arrêter* peut ne pas

être indiquée sur les cartes d'extension (en effet, cette intention étant atteignable de toutes les intentions génériques de la carte et avec la même stratégie, son écriture peut être considérée comme facultative).

Pour chaque ensemble de patrons d'extensions réalisant l'intégration d'un même concept, on extrait plusieurs points spécifiques de leurs descripteurs. Tout d'abord, tous ces patrons s'organisent selon une intention générique (qui se spécialise par les stratégies possibles). Cette intention peut donc être filtrée et correspondra à une intention de la carte d'extension.

Tous les patrons définis pour une extension similaire (intégration du même concept) permettent de définir une intention générique de la carte d'extension (un nœud du graphe).

Il est alors nécessaire de définir les arcs permettant de naviguer entre les intentions. Ceci se fait en étudiant chacun d'entre eux l'un par rapport aux autres pour déterminer quels sont ceux qui doivent impérativement être exécutés dans un ordre spécifique, donc qui déterminent des *contraintes de précédence*, et ceux faisant parti d'un même ordre logique, donc qui déterminent des *contraintes de complétude*. Ces deux ensembles permettent de définir la majorité des arcs du graphe.

Les contraintes de précédence permettent de donner l'ordre obligatoire d'exécution entre certaines intentions.

Chaque contrainte de précédence d'un patron d'une intention cible permet de définir l'intention source d'une ou plusieurs de ses sections.

D'une façon un peu différente, les contraintes de complétude permettent, non pas de définir un ordre péremptoire d'exécution, mais plutôt un ordre logique permettant une certaine complétude de la méthode étendue.

Chaque contrainte de complétude d'un patron d'une intention source permet de définir l'intention cible d'une ou plusieurs de ses sections.

L'absence de contrainte de précédence donne à l'ingénieur de méthodes la possibilité de commencer l'exécution de la carte par cette intention, sans pour cela mettre en danger la cohérence de la méthode obtenue. L'intention source de ces sections sera donc l'intention *Démarrer*.

Si les patrons d'une intention cible ne possèdent pas de contrainte de précédence, c'est que ses sections auront l'intention *Démarrer* comme intention source.

Finalement, toutes les spécialisations de l'intention générique correspondent aux différentes stratégies que l'on peut appliquer pour obtenir la réalisation de cette intention générique. Ces stratégies correspondront à autant de sections allant d'une même intention source à cette intention générique cible.

Les stratégies applicables pour réaliser une intention générique définissent l'ensemble des sections (sous forme de cluster) où cette intention est intention cible. Il est à noter que le cluster regroupera autant de sections qu'il y avait de patrons d'extensions dans cet ensemble.

La carte d'extension permet à celui qui l'exécute d'étendre sa méthode de façon complète mais lui permet également de ne l'étendre que partiellement, pour un besoin spécifique. Chaque réalisation d'une intention donnera donc la possibilité à l'ingénieur d'arrêter l'exécution de la carte.

Toute intention est intention source d'une section dont l'intention cible est l'intention *Arrêter* et la stratégie *Stratégie d'arrêt*. Par souci de clarté et de simplicité lors de la lecture des cartes, l'intention cible *Arrêter* n'est pas toujours indiquée, elle est cependant systématiquement atteignable de toute intention source de la carte.

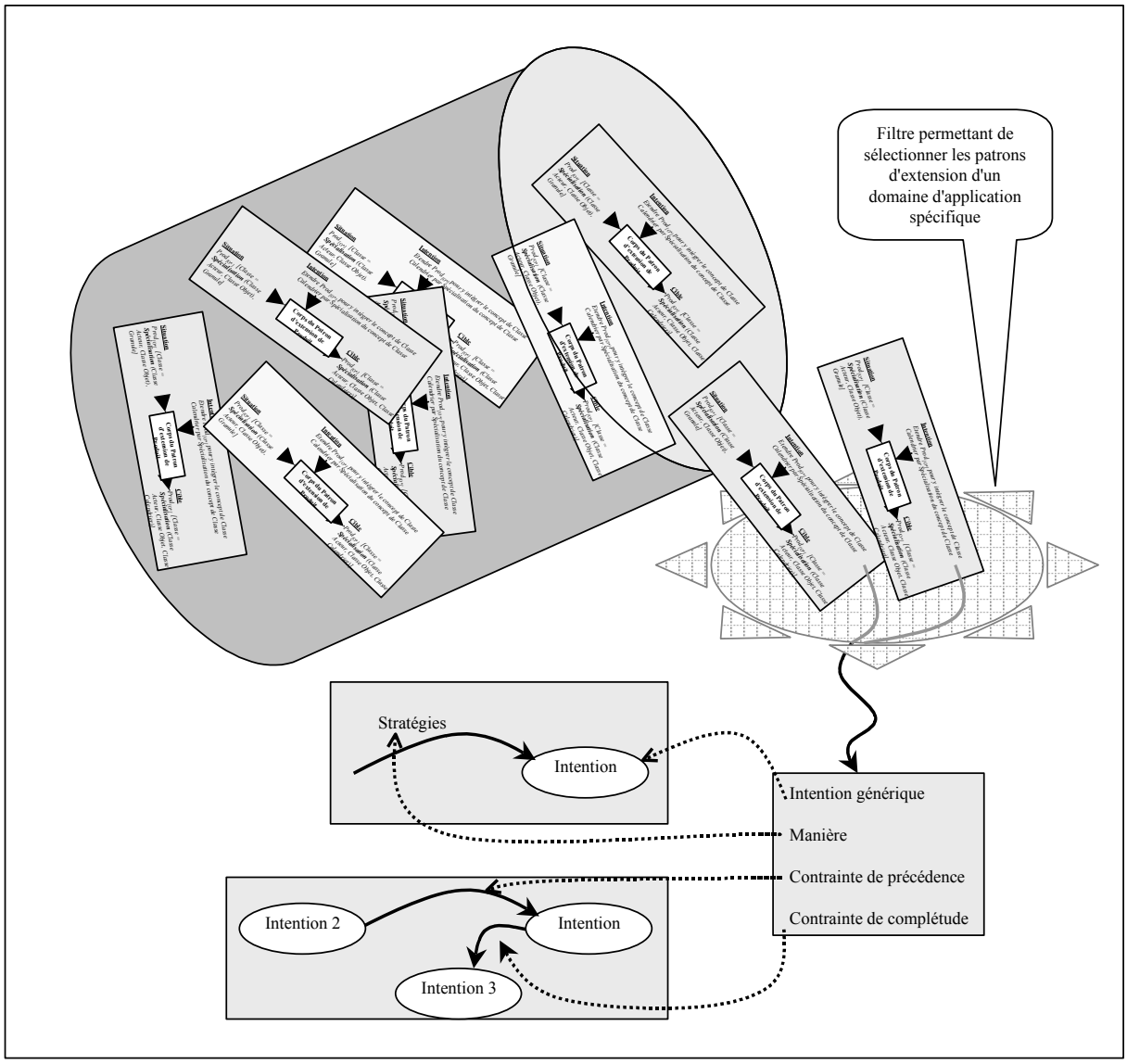


Figure 74: Illustration des règles de passage

14.2 Exemple de définition d'une carte d'extension

Prenons l'exemple de la définition de la carte d'extension d'une méthode aux applications temporelles (pour des raisons de clarté, seul un petit nombre de patrons sont étudiés dans cet exemple ; pour plus de détails sur la carte d'extension aux applications temporelles, l'annexe B décrit l'ensemble des patrons possibles).

L'application de la 0 permet d'extraire l'ensemble des patrons d'extension de la bibliothèque. Les intentions de ces patrons sont listées dans la Figure 75.

Etendre $Prod_{[M]}$ par SPECIALISATION pour intégrer les référentiels temporels.
Etendre $Prod_{[M]}$ par GENERALISATION pour intégrer les référentiels temporels.
Etendre $Proc_{[M]}$ par stratégie SEQUENTIELLE GLOBALE pour intégrer la construction des référentiels temporels.
Etendre $Proc_{[M]}$ par stratégie INTEGREE pour intégrer la construction des référentiels temporels.
Etendre $Prod_{[M]}$ par REMPLACEMENT pour intégrer les domaines temporels.
Etendre $Prod_{[M]}$ par GENERALISATION pour intégrer les domaines temporels.
Etendre $Prod_{[M]}$ par COMPOSITION pour intégrer les domaines temporels.
Etendre $Proc_{[M]}$ par stratégie SEQUENTIELLE LOCALE pour intégrer la construction des domaines temporels.
Etendre $Proc_{[M]}$ par stratégie INTEGREE pour intégrer la construction des domaines temporels.

Figure 75: Intentions des patrons pour le domaine d'applications temporelles

L'application de la 0 permet de définir les deux premières intentions de la carte alors que l'application de la 0 permet de définir les intentions génériques présentes dans la carte d'extension correspondante au domaine des applications temporelles, ainsi que l'illustre la Figure 76.

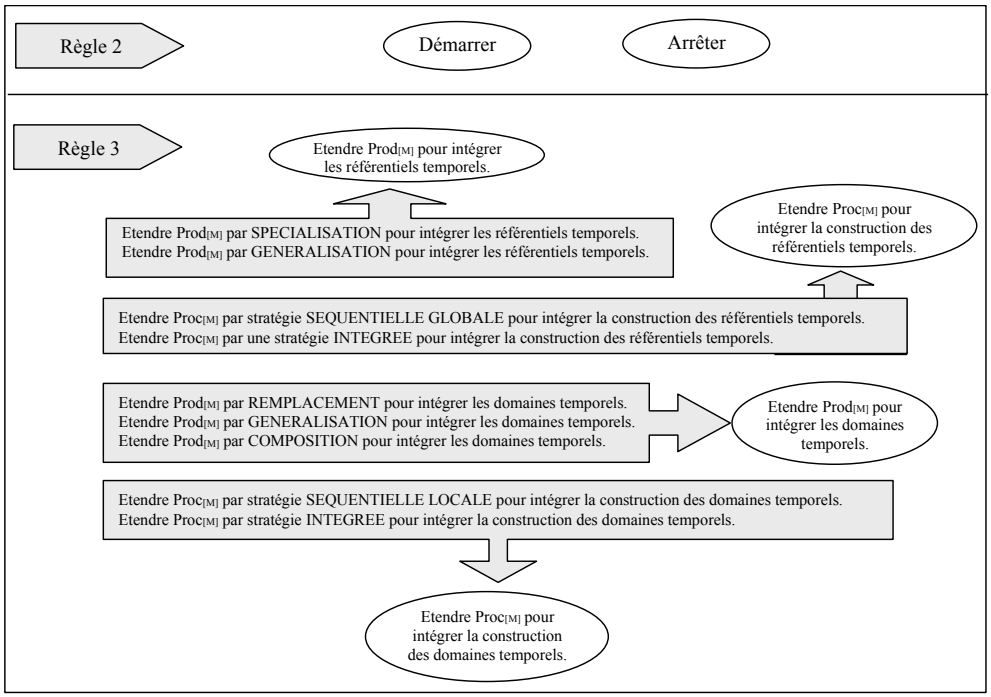


Figure 76: Nœuds de la carte d'extension

L'application des règles 4,5 et 6 permet de définir dans quel ordre doivent s'exécuter les patrons.

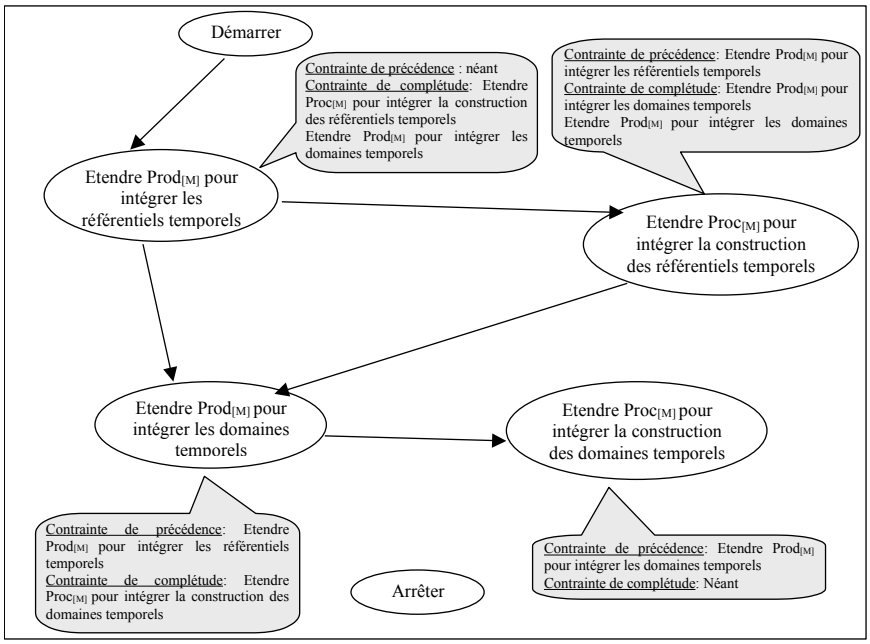


Figure 77: Arcs de la carte d'extension

L'application de la 0 permet de définir les arcs des sections en définissant les stratégies possibles pour aller d'une intention source à une intention cible.

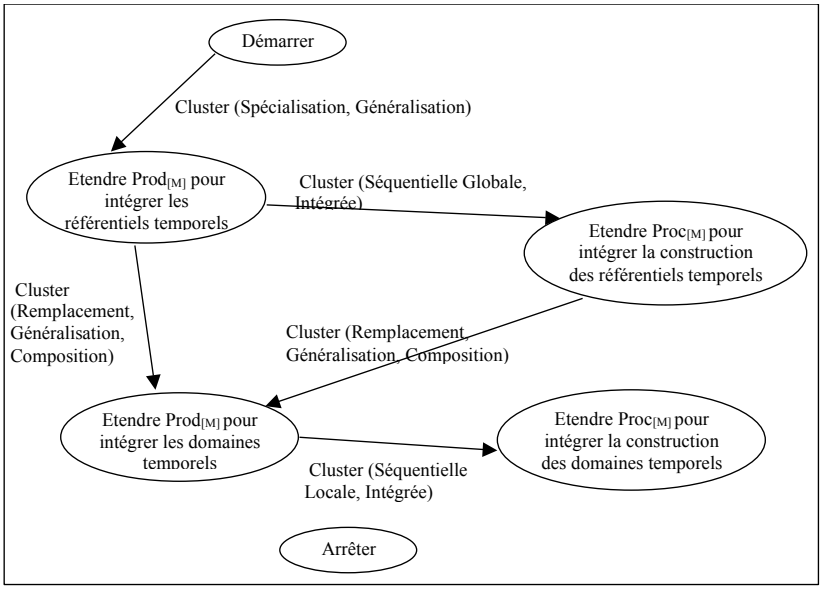


Figure 78: Insertion des stratégies d'extension

L'application de la 0 permet de donner à l'ingénieur de méthodes toute latitude pour arrêter l'exécution de la carte dès la réalisation d'une intention accomplie.

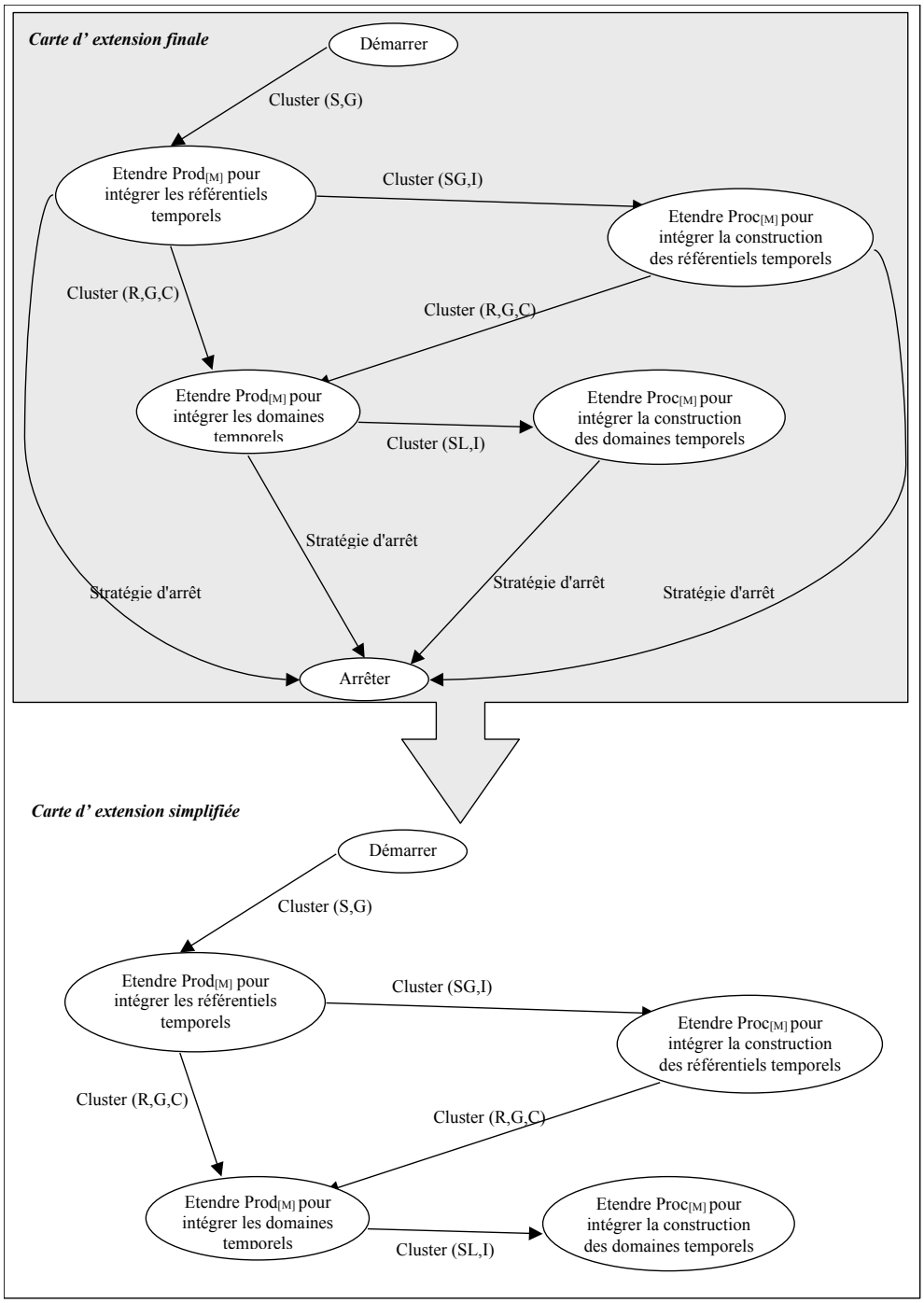


Figure 79: Carte d'extension aux applications temporelles

Chapitre VI : Construction des patrons d'extension

Introduction

Dans le chapitre précédent, il a été défini qu'un patron d'extension est représenté dans une carte d'extension par une section spécifique. Chacune de ces sections (c.-à-d. chaque patron) représente la façon d'atteindre une intention cible, à partir d'un nœud dans la carte, par l'exécution d'une stratégie particulière. La figure suivante illustre l'équivalence entre le concept de *Section* et celui de *Patron* sur une représentation de deux sections d'une carte d'extension.

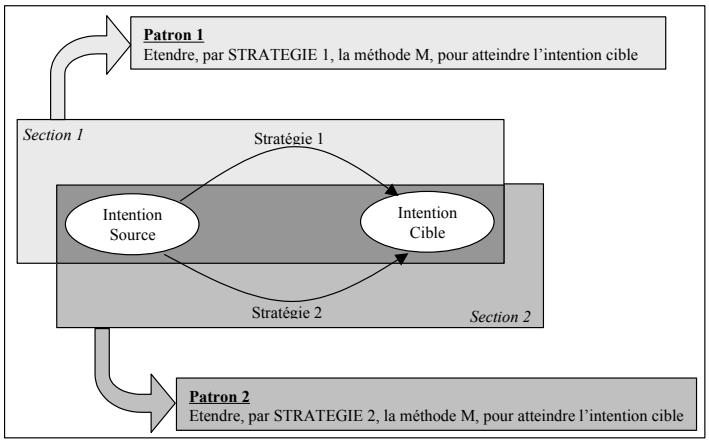


Figure 80: Equivalence entre le concept de section et celui de patron

Il est possible de constater, grâce à la figure précédente, que le patron à appliquer entre deux intentions sera différent selon la stratégie utilisée. On peut donc dire qu'il y aura autant de patrons d'extension différents applicables, en partant d'une intention source pour atteindre une intention cible, que de stratégies.

La problématique de l'extension d'une méthode, pour un domaine d'application donné, commence par l'inventaire des éléments à intégrer dans cette méthode (c.-à-d. l'ensemble des intentions de la carte d'extension associée), puis l'ingénieur de méthodes doit effectuer l'inventaire des stratégies applicables pour réaliser ces intentions et, finalement, chacune des sections identifiées doit être décrite par un patron d'extension.

Le problème qui se pose ici est que la réalisation de l'inventaire de toutes les stratégies possibles présentes dans une carte est un travail long, difficile et surtout stratégique. En effet, si l'ingénieur de méthodes omet une section dans une carte d'extension, il suffit que le processus d'extension d'une méthode ait besoin de cette section pour atteindre son objectif pour que cette extension ne puisse se dérouler. Ce mémoire propose donc un ensemble de stratégies génériques dont la réalisation est paramétrable selon la méthode à étendre et l'élément à intégrer. Le fait d'appliquer toutes ces stratégies génériques à toutes les intentions de la carte permet donc de garantir la complétude des stratégies présentes dans la carte d'extension.

De plus, pour faciliter la réalisation des sections de la carte d'extension (c.-à-d. la description des patrons d'extensions associés), toute la connaissance relative à la construction de patrons d'extension selon une stratégie générique est encapsulée dans un méta-patron spécifique. L'ingénieur de méthodes,

une fois les éléments à intégrer identifiés, pourra donc directement réaliser les intentions de la carte par l'application de tous les méta-patrons. En effet, chaque application d'un méta-patron entraînera la construction d'un patron d'extension permettant de réaliser une intention cible, en partant d'une intention source, par l'application d'une stratégie spécifique. Ce processus de construction permet donc à la fois de guider l'ingénieur de méthodes lors de l'identification des sections de la carte mais également lors de la construction des patrons associés.

L'exemple illustré par la figure suivante visualise l'instantiation du méta-patron d'extension utilisant la stratégie générique SPECIALISATION pour la construction de trois patrons d'extensions utilisant cette stratégie et dont chacun est représenté sous forme de section dans une carte d'extension.

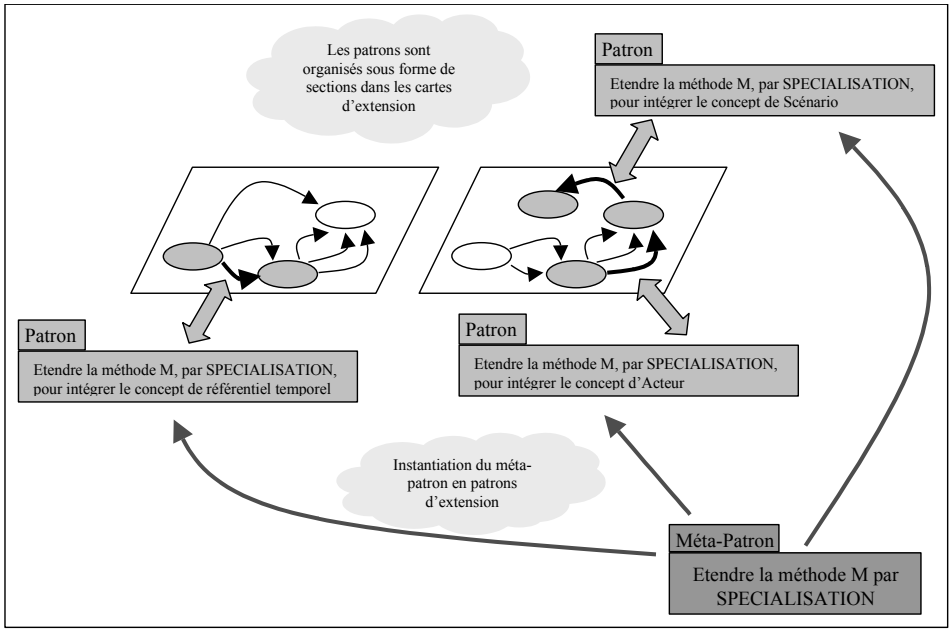


Figure 81: Instantiation des méta-patrons en patrons d'extension

La description complète des méta-patrons est fournie à l'annexe A. Cette section se focalise donc surtout sur les stratégies d'extension existantes et leurs illustrations.

15 Stratégies d'extension

Les opérateurs de modification de la méthode (cf. chapitre IV) détaillés dans le corps d'un patron d'extension sont définis suivant les méta-modèles de **PRODUIT** et de **DÉMARCHE**. Certaines techniques, appelées des *stratégies d'extension*, ont donc été définies pour formaliser certains ensembles d'opérateurs et obtenir des modifications génériques.

15.1 Formalisme d'utilisation des stratégies d'extension

Les stratégies d'extension permettent de modifier une méthode. La Figure 82 illustre le formalisme d'application de ces stratégies. Chacune prend en compte trois paramètres représentant :

l'élément de la méthode qui va être modifié - [Elément X] (cet élément fait partie de la méthode avant extension),
l'élément que l'ingénieur de méthodes souhaite intégrer - [Elément Y] (cet élément fait partie de la méthode après extension) et
le possible élément rendu inutile du fait de l'extension – [Elément D].

```
Stratégie ([Elément X], [Elément Y], [Elément D])
```

Figure 82: Structure de l'utilisation d'une stratégie d'extension

Il est à noter que les deux premiers arguments sont obligatoires alors que le troisième – l'[Elément D] – ne l'est pas. En effet, toute extension ne modifie pas la méthode d'origine au point de rendre caduc l'un de ses éléments mais, si tel est le cas, il est nécessaire de pouvoir supprimer cet élément pour permettre de conserver la cohérence de la méthode.

15.2 Exemple de stratégie d'extension

Prenons l'exemple d'un patron d'extension permettant d'intégrer le concept de *Classe Acteur* dans une méthode possédant déjà le concept de *Classe* spécialisé en *Classe Objet* et *Classe Horloge*. La meilleure manière d'insérer ce nouveau concept est d'ajouter une spécialisation au concept de *Classe* pour représenter celui de *Classe Acteur*.

La situation de la méthode avant l'exécution de la stratégie sera donc la suivante (avec le langage de description du **PRODUIT** décrit dans le chapitre IV).

```
Classe = Spécialisation (Classe Objet, Classe Horloge)
```

Figure 83: Description d'une partie de la méthode avant l'application de la stratégie

[Elément X] représente donc ici le concept de *Classe* et [Elément Y] celui de *Classe Acteur*. Nous pensons qu'il serait judicieux d'utiliser une spécialisation donc la stratégie d'extension à utiliser sera la SPECIALISATION. Prenons le cas où cette modification n'entraîne pas l'inutilité d'un autre élément déjà présent dans la méthode ([Elément D] = ϕ). La structure de l'utilisation de cette stratégie dans cet exemple précis sera donc représentée de la façon suivante.

```
SPECIALISATION (Classe, Classe Acteur, ϕ)
```

Figure 84: Exemple d'utilisation de la stratégie d'extension SPECIALISATION

La situation de la méthode après l'exécution de la stratégie sera donc une méthode étendue dont une partie est indiquée à la figure suivante (avec le langage de description du **PRODUIT** décrit au chapitre III).

```
Classe = Spécialisation (Classe Objet, Classe Horloge, Classe Acteur)
```

Figure 85: Description d'une partie de la méthode après l'application de la stratégie

La connaissance relative aux stratégies est encapsulée dans un type de patron spécifique appelé Méta-patron. Le chapitre précédent a introduit la description des patrons d'extension. Les méta-patrons suivent le même type de description.

16 Description d'un méta-patron

Comme les patrons d'extension, les méta-patrons sont définis suivant la description indiquée dans le chapitre précédent. Ils sont donc composés des deux parties correspondant à leur signature et à leur corps.

16.1 Signature d'un méta-patron

La signature d'un méta-patron ne comprend que la partie formelle de sa description, c.-à-d. l'interface définissant sa situation, son intention et sa cible.

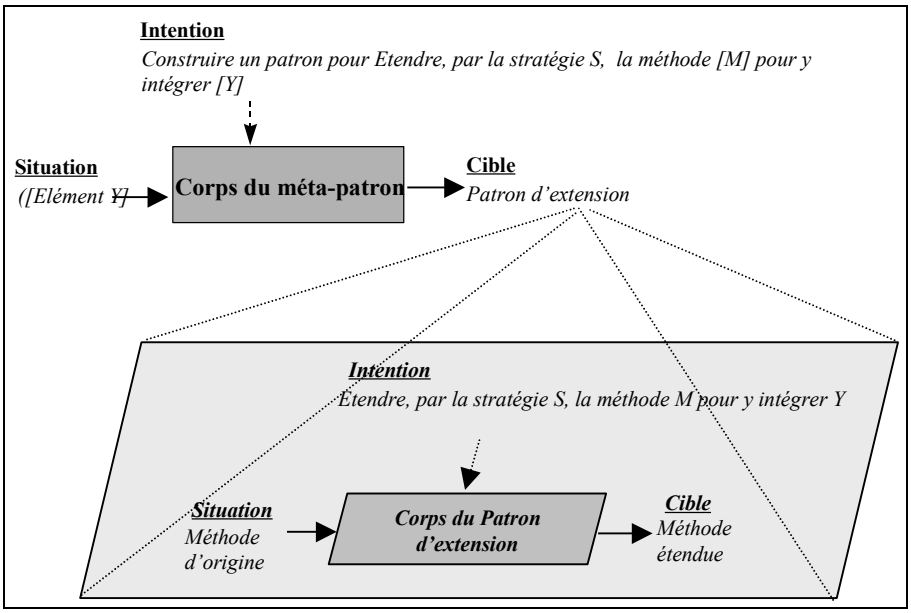


Figure 86: Interface d'un méta-patron

La situation d'un méta-patron correspond à l'élément Y, élément dont on veut agrémenter la méthode d'origine.

L'intention d'un méta-patron est de construire un patron d'extension qui permettra d'étendre la méthode d'origine pour y intégrer cet élément en appliquant une certaine stratégie.

La cible d'un méta-patron est donc un patron d'extension pour un élément défini et suivant une certaine stratégie d'extension.

16.2 Corps d'un méta-patron

Le corps d'un méta-patron correspond aux opérations permettant de construire ce patron d'extension, c.-à-d. de définir sa situation, son intention, sa cible ainsi que son corps formé de la séquence

d'opérateurs correspondant à la stratégie. En effet, chaque stratégie correspond à une suite d'opérateurs de modification contenant trois inconnues : l'élément X, l'élément Y et l'élément D (les trois arguments de la stratégie). L'élément Y est connu lors de l'application d'un méta-patron pour la création d'un patron d'extension alors que les éléments X et D ne sont connus que lors de l'instanciation de ce patron d'extension à une méthode d'origine donnée.

16.3 Typologie de méta-patrons

De même que pour les patrons d'extension, les méta-patrons peuvent être différenciés selon deux types spécifiques : ceux permettant de construire un patron d'extension de **PRODUIT** et ceux permettant de construire un patron d'extension de **DÉMARCHE**.

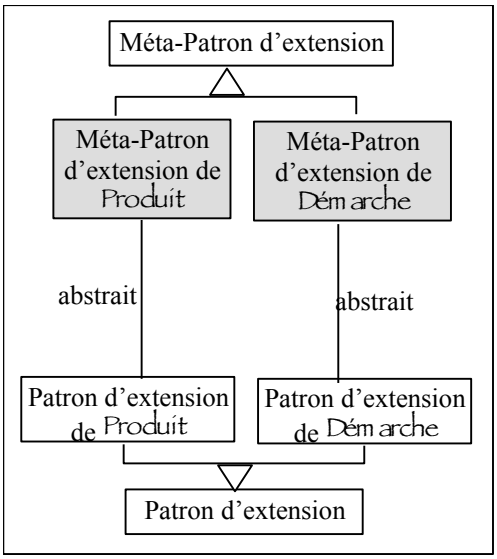


Figure 87: Typologie des méta-patrons d'extension de méthodes

Les particularités spécifiques des méta-patrons de **PRODUIT** et de **DÉMARCHE** sont expliquées en détail dans les sections suivantes.

16.4 Exemple d'application d'un méta-patron

La figure suivante illustre les instantiations successives permettant de passer d'un méta-patron au patron d'extension à un concept spécifique puis au patron d'extension permettant d'étendre une méthode spécifique. Ici, le concept à intégrer est le concept de *Classe Calendrier* et la méthode est O*.

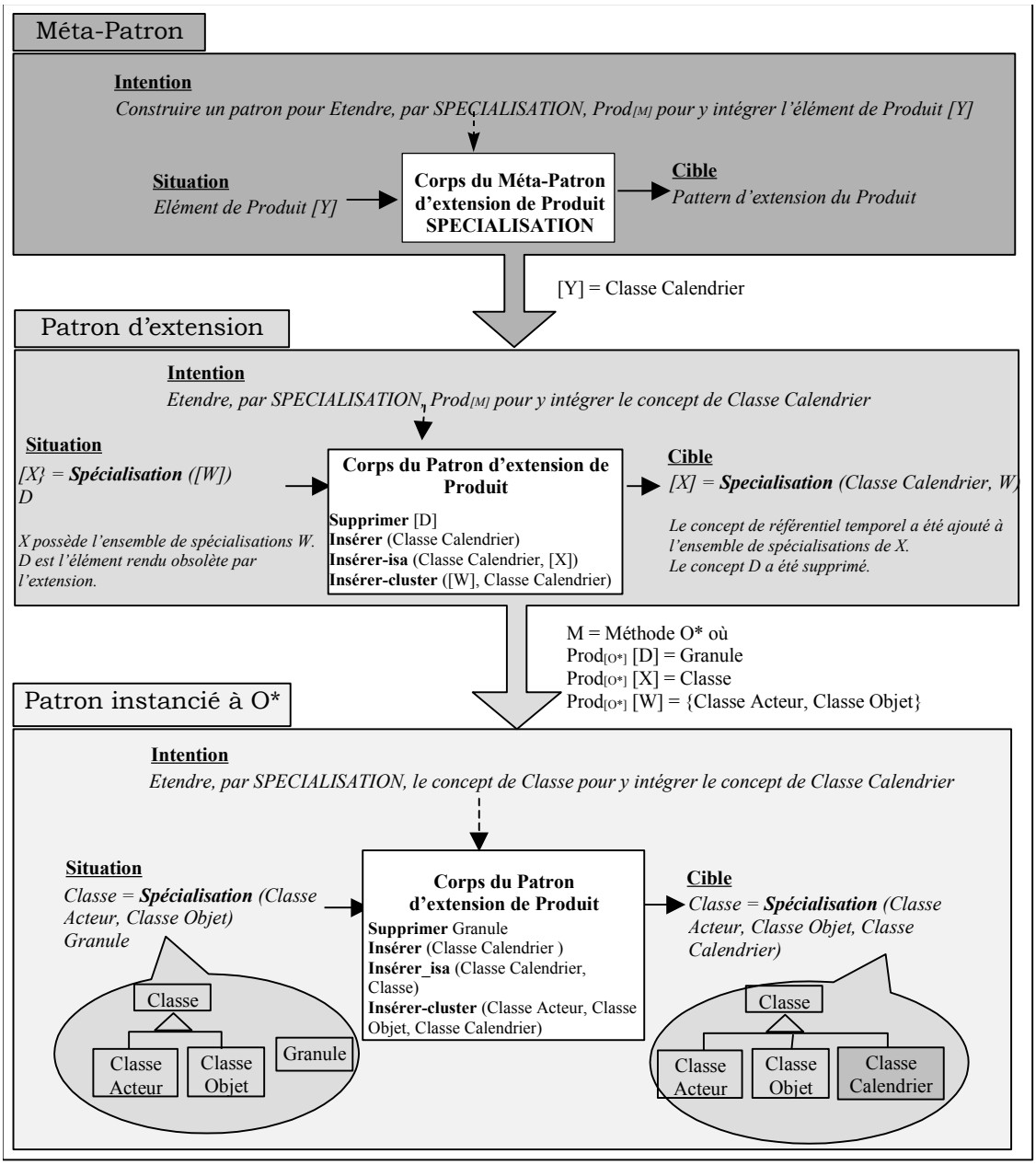


Figure 88: Exemple d'instantiation d'un méta-patron

Cette figure illustre les deux étapes permettant de créer un patron d'extension spécifique à une méthode. La première étape représente l'instantiation du méta-patron d'extension par SPECIALISATION pour le concept de *Classe Calendrier*. Cette opération permet la construction du patron d'extension permettant d'intégrer ce concept en utilisant la stratégie d'extension par spécialisation. La deuxième étape représente l'instantiation de ce patron d'extension à la méthode O* pour permettre l'extension de celle-ci avec le concept de *Classe Calendrier*.

L'objectif de cette section étant la construction des patrons d'extension, nous nous focalisons donc essentiellement sur la définition de tous les méta-patrons possibles par l'inventaire des stratégies d'extension applicables.

17 Spécification d'un Méta-patron d'extension de PRODUIT

Lorsque la partie **PRODUIT** d'une méthode ne convient pas à l'ingénieur de méthodes, celui-ci peut décider d'étendre le **PRODUIT** par le biais de l'application d'un patron d'extension du **PRODUIT** défini dans sa bibliothèque de patrons. Cette bibliothèque est peuplée de façon générique par le biais de patrons particuliers appelés *méta-patrons de **PRODUIT***. Ces méta-patrons permettent à l'ingénieur de méthodes de construire des patrons d'extension pertinents adaptés au domaine d'application en cours. Ces patrons peuvent alors être appliqués à la méthode et celle-ci possédera les concepts nécessaires pour permettre à l'ingénieur de construire son application de façon complète et cohérente.

17.1 Interface d'un Méta-patron d'extension de PRODUIT

Un méta-patron de **PRODUIT** peut être décrit grâce à son interface, ainsi que l'illustre la Figure 89.

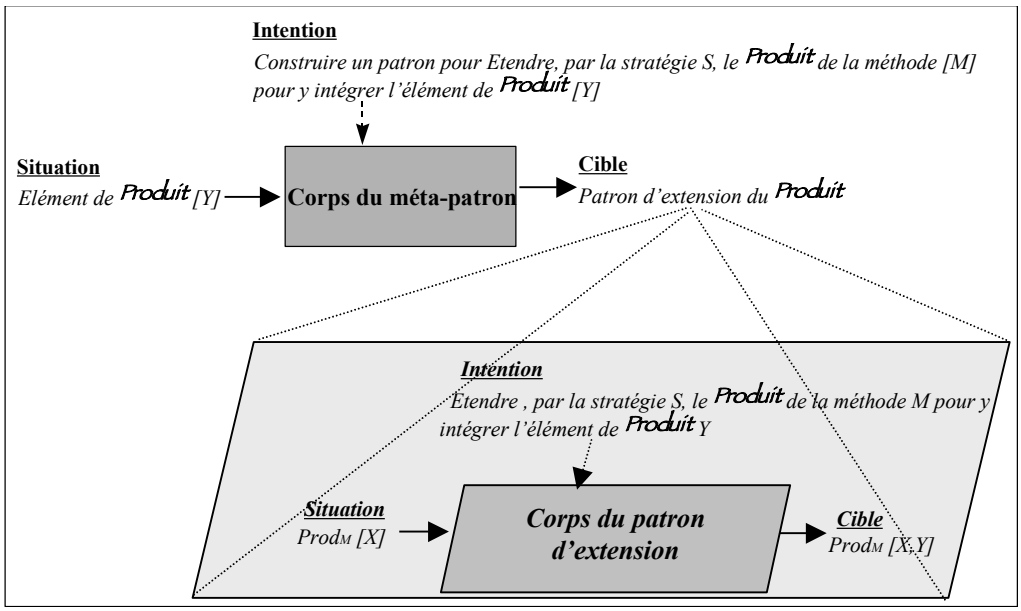


Figure 89: Interface d'un méta-patron de PRODUIT

La situation d'un méta-patron de **PRODUIT** correspond à l'élément de **PRODUIT** dont on veut agrémenter la méthode d'origine.

L'intention d'un méta-patron de **PRODUIT** correspond au fait de construire un patron pour étendre le **PRODUIT** de la méthode d'origine pour y intégrer cet élément de **PRODUIT** en appliquant une certaine stratégie.

La cible d'un méta-patron de **PRODUIT** sera en fait un patron d'extension du **PRODUIT** pour un élément de **PRODUIT** défini et suivant une certaine stratégie d'extension.

Le corps d'un méta-patron correspond aux opérations de création du patron d'extension de **PRODUIT**. Le corps de ce patron est composé d'une suite d'opérateurs de transformation correspondant à la stratégie exécutée pour satisfaire l'extension.

Les stratégies d'extension de la partie **PRODUIT** d'une méthode sont décrites dans la section suivante.

17.2 Stratégies d'extension du PRODUIT d'une méthode

Le chapitre III définit une extension de méthode comme *une technique permettant d'englober un plus grand nombre de choses dans l'ensemble de départ*. Une extension de la partie **PRODUIT** d'une méthode revient donc à introduire de nouveaux concepts dans les modèles existants.

Cette introduction peut se faire de deux manières différentes : soit le concept que l'ingénieur de méthodes souhaite intégrer dans la partie **PRODUIT** de la méthode d'origine est un élément totalement nouveau, soit c'est un concept qui est déjà représenté dans la méthode mais de façon incomplète ou trop incorrecte pour satisfaire ses besoins.

Introduction d'un concept totalement nouveau dans la méthode d'origine

L'introduction de ces nouveaux concepts doit également s'accompagner d'un rattachement de ceux-ci au reste du modèle. En effet, tout concept n'a de sens que s'il possède des liens avec les autres concepts déjà présents dans la méthode. Ces greffes s'effectuent en instanciant les différents types des liens possibles définis dans le méta-modèle de cette partie **PRODUIT**. Il est utile de remarquer que le processus d'extension sera différent selon ces derniers. Ceci nous permet de dire que les extensions peuvent être définies selon le type de lien permettant de relier le nouveau concept au modèle existant. La Figure 90 illustre les différents liens définis dans le méta-modèle générique de la partie **PRODUIT** des méthodes prises en compte dans ce mémoire.

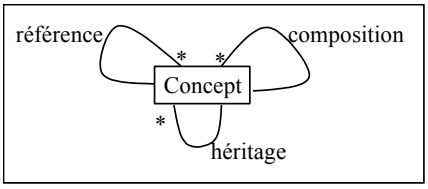


Figure 90 : Méta-modèle générique de la partie PRODUIT des méthodes orientées-objet

Un nouveau concept peut donc être inséré dans le modèle de **PRODUIT** d'une méthode existante en le reliant à un autre concept par le biais d'un lien d'héritage, de composition ou de référence.

Lien d'héritage

L'ingénieur de méthodes peut insérer un concept dans la partie **PRODUIT** d'origine en le rattachant aux concepts existant avec un lien d'héritage. Le nouveau concept peut alors être incorporé dans le modèle comme spécialisation d'un concept soit existant, soit inexistant.

Dans le premier cas, il suffit d'ajouter le concept et le lien d'héritage entre le concept existant et le nouveau concept. Cette technique s'applique si la méthode d'origine comporte préalablement un

concept pouvant être spécialisé avec le nouveau concept. Prenons l'exemple de la Figure 82 concernant la partie **PRODUIT** de la méthode OMT qui possède le concept d'*Evénement*. L'insertion du nouveau concept d'*Evénement Temporel* peut se faire par spécialisation du concept d'*Evénement* en *Evénement Temporel*. L'extension peut donc s'effectuer en insérant le nouveau concept puis le lien de spécialisation.

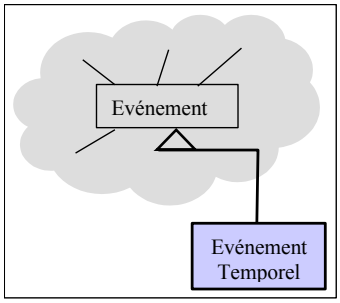


Figure 91: Exemple d'insertion par le biais d'un lien d'héritage (Cas 1)

Dans le second cas, au contraire, la méthode d'origine ne comporte pas de concept permettant de prendre le nouveau concept comme spécialisation mais possède un concept ayant une sémantique très similaire à celui-ci. Dans ce cas très particulier, l'extension permet d'insérer un autre concept permettant de généraliser à la fois le concept déjà existant et le nouveau concept que l'on souhaite intégrer. Considérons de nouveau l'exemple de la méthode OMT. Cette méthode possède le concept de *Classe Objet*. L'insertion du concept de *Classe Acteur* peut se faire avec cette technique puisque ces deux concepts sont très proches sémantiquement. On insère donc une généralisation du concept de *Classe Objet* que l'on appelle *Classe*, puis on intègre le concept de *Classe Acteur* qui est donc rattaché à ce nouveau concept comme spécialisation ainsi que l'indique la Figure 92.

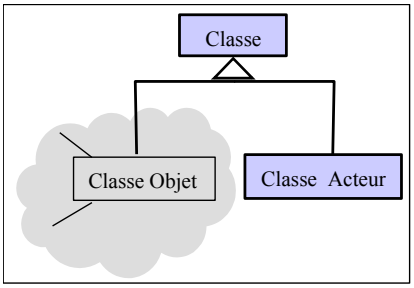


Figure 92: Exemple d'insertion par le biais d'un lien d'héritage (Cas 2)

Insérer un concept par le biais d'un lien d'héritage revient donc à utiliser un lien de spécialisation (le concept inséré devient un élément spécialisé d'un élément existant) ou de généralisation (le concept inséré et un élément existant deviennent spécialisations d'un nouvel élément généralisé). Chacune de ces deux différentes façons d'insérer un nouvel élément dans une méthode existante représente donc une stratégie particulière d'extension du **PRODUIT** d'une méthode, la SPECIALISATION et la GENERALISATION.

Lien de composition

L'ingénieur de méthodes peut également insérer un nouveau concept dans la partie **PRODUIT** d'une méthode par le biais d'un lien de composition. Le nouveau concept peut alors être inséré dans le

modèle de la méthode d'origine comme un élément composé, ou composant, d'un concept existant déjà préalablement dans la méthode.

Dans le premier cas, le concept est inséré dans le modèle de la méthode puis rattaché aux autres concepts comme composant d'un concept déjà existant. Prenons le cas de la méthode OMT contenant le concept de *Classe Objet*. Ce concept est lui-même composé de ceux de *Propriété*, *Evénement* et *Opération*. L'ingénieur de méthodes peut étendre cette méthode en y intégrant le concept de *Contrainte de Classe Objet*. La meilleure manière d'intégrer ce concept est d'en faire un composant de celui de *Classe Objet*, au même titre que les trois autres concepts déjà existants. L'extension intégrera donc le concept de *Contrainte* dans la composition du concept de *Classe Objet*, ainsi qu'il est indiqué dans la Figure 93.

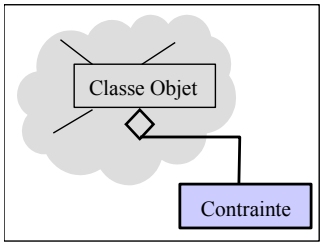


Figure 93: Exemple d'insertion par le biais d'un lien de composition (Cas 1)

A l'opposé, dans le deuxième cas, le concept que l'ingénieur de méthodes souhaite intégrer dans la méthode devient, non pas un composant d'un concept existant, mais un concept composé d'un concept existant. Par exemple, prenons le cas d'une méthode possédant le concept d'*Evénement Externe* ainsi que le visualise la Figure 94. L'ingénieur de méthodes exécute une extension sur cette méthode pour y intégrer le concept de *Classe Acteur*. La meilleure manière d'insérer ce concept est de le rattacher à celui d'*Evénement Externe* par le biais d'un lien de composition, avec comme composé le concept de *Classe Acteur*, et comme composant celui d'*Evénement Externe*.

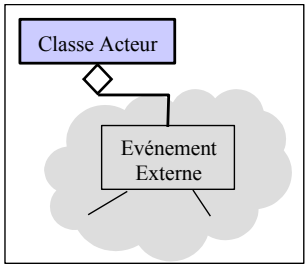


Figure 94: Exemple d'insertion par le biais d'un lien de composition (Cas 2)

Insérer un concept par le biais de ce lien est donc caractérisé par l'utilisation d'un lien de composition (le concept inséré devient un élément composant d'un élément existant) ou de décomposition (le concept inséré devient un élément composé d'un élément existant). Chacune de ces deux différentes façons d'insérer un nouvel élément dans une méthode existante représente donc une stratégie particulière d'extension du **PRODUIT** d'une méthode, la COMPOSITION et la DECOMPOSITION.

Lien de référence

Un autre lien présent dans le méta-modèle générique des méthodes orientées objet est le lien de référence. L'ingénieur de méthodes peut donc insérer un nouveau concept dans la méthode d'origine par le biais de ce type de lien. Dans ce cas, il lui suffit d'intégrer le concept puis son lien avec un concept existant. Prenons l'exemple d'une méthode possédant le concept d'*Action*. Le rattachement du nouveau concept d'*Agent* peut très bien se faire par un lien de référence entre ces deux concepts (Une *Action* est effectuée par un *Agent* et un *Agent* effectue une ou plusieurs *Action*(s)).

Insérer un concept par le biais d'un lien de référence caractérise donc le fait que le concept inséré référence, ou est référencé par, un élément déjà existant dans la méthode d'origine. Cette façon d'insérer un nouvel élément dans une méthode existante représente donc une stratégie particulière d'extension du **PRODUIT** d'une méthode, la REFERENCE.

Introduction d'un concept représenté de façon incomplète dans la méthode d'origine

En dehors des insertions de nouveaux concepts reliés au modèle par les liens définis dans le méta-modèle, il est également possible d'étendre une méthode, non pas en insérant un lien, mais en remplaçant l'un des concepts de la méthode pour en changer la description. Prenons l'exemple d'une méthode possédant le concept de *Granule*. L'ingénieur de méthodes souhaite peaufiner la notion de granule en intégrant le concept de *Calendrier*. Le concept particulier *Granule* devient donc un concept qui n'est pas assez poussé pour les besoins de l'ingénieur de méthodes. Il peut donc remplacer celui-ci par le concept beaucoup plus défini de *Calendrier*.

Etendre une méthode en insérant un élément par cette technique est donc caractérisée par l'utilisation d'un remplacement de concept. Cette manière d'intégrer un concept dans une méthode représente donc une stratégie particulière d'extension du **PRODUIT** d'une méthode, le REMPLACEMENT.

Pour résumer, nous pouvons énumérer la liste suivante de stratégies d'extension du **PRODUIT** possibles dans le domaine des applications orientées objets (Chacune de ces stratégies sera supportée par un Méta-patron de **PRODUIT**).

17.2.1 SPECIALISATION

L'application de la stratégie SPECIALISATION permet une extension de la méthode par le biais d'un lien de spécialisation.

17.2.1.1 Principe d'application de la stratégie SPECIALISATION

L'élément de **PRODUIT** peut être intégré comme une spécialisation d'un élément de **PRODUIT** existant plus générique. Cela signifie que l'on va ajouter le concept qui nous intéresse ([Élément Y]) et le relier au reste du modèle par le biais d'un lien de spécialisation avec un élément déjà existant ([Élément X]).

La recherche d'une structure générique permettant d'effectuer ce traitement nous a conduit à l'élaboration des opérateurs suivants : *{**Insérer** [Élément Y], **Insérer-isa** ([Élément Y], [Elément X])}*

où *[Élément Y]* représente le concept spécialisé à intégrer et *[Élément X]* le concept généralisé déjà présent. Il est également utile d'insérer le nouvel élément dans un nouveau cluster qui sera donc formé de cet élément et d'autres spécialisations de l'élément que l'on étend, ce qui se formalise par l'opérateur ***Insérer-cluster*** *([Ensemble W], [Elément Y])* où *[Elément Y]* est l'élément que l'on intègre et *[Ensemble W]* l'ensemble des concepts spécialisés devant faire partie de ce cluster.

Il est à noter que, dans certains cas, l'élément de **PRODUIT** que l'on intègre remplace en fait un élément déjà présent dans le modèle. C'est à dire que cet élément d'origine ne correspond pas à ce que l'ingénieur de méthodes voudrait utiliser et qu'il décide de le remplacer complètement par ce nouvel élément que l'on intègre par spécialisation. Cette opération se formalise avec l'opérateur suivant: ***Supprimer*** *[Élément D],* où *[Élément D]* représente l'élément de la méthode d'origine que l'on souhaite remplacer.

La figure suivante résume le principe d'application de cette stratégie de façon graphique et textuelle.

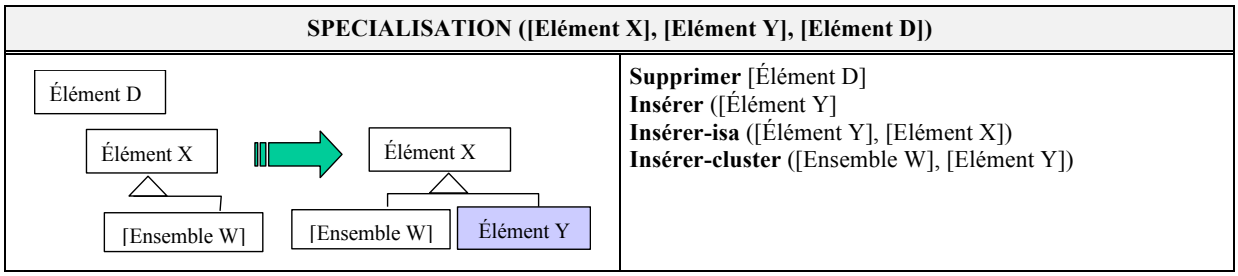


Figure 95: Principe d'application de la stratégie SPECIALISATION

17.2.1.2 Exemples d'application de la stratégie SPECIALISATION

La Figure 96 illustre trois exemples d'application de cette stratégie sur trois différentes parties de méthodes.

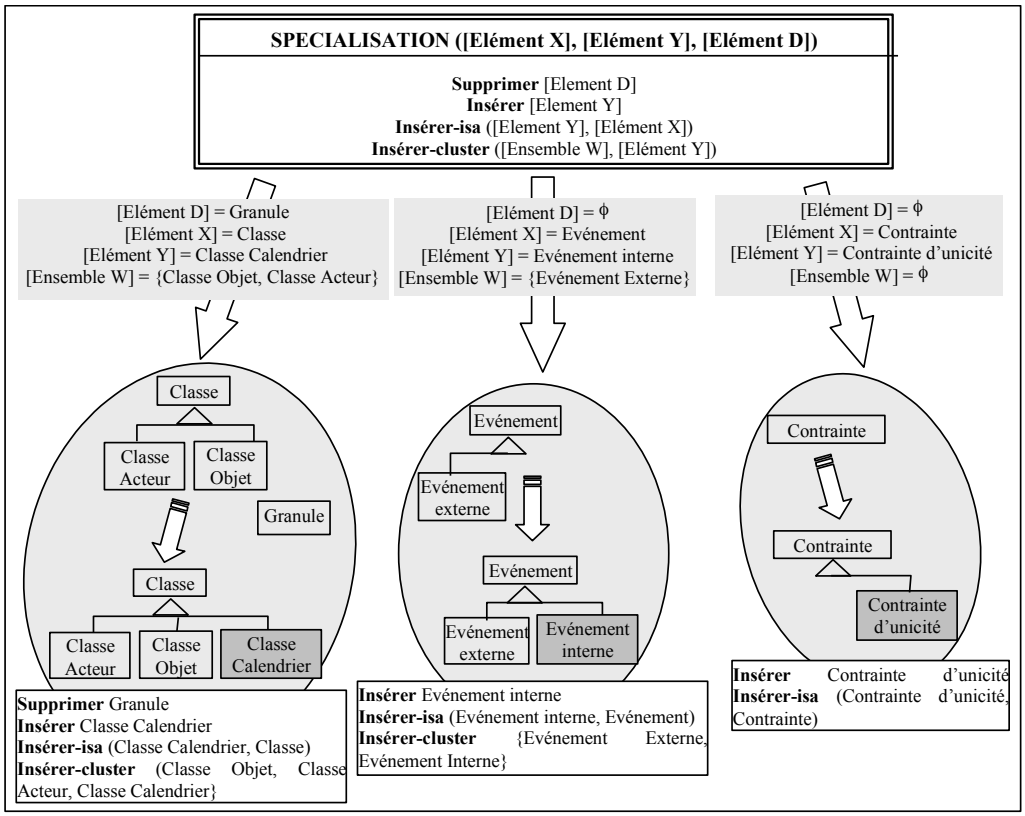


Figure 96: Exemples d'application de la stratégie SPECIALISATION

Le premier exemple modifie une partie de méthode contenant le concept de *Classe* spécialisable en *Classe Acteur* ou *Classe Objet*, ainsi qu'un concept de *Granule*. L'application de la stratégie permet d'intégrer une nouvelle spécialisation au concept de *Classe* : la *Classe Calendrier*. Le concept de *Granule*, rendu inutile puisque redéfini dans le concept de *Classe Calendrier* est également supprimé de la méthode.

Dans le deuxième exemple, le concept d'*Evénement* est déjà spécialisable en *Evénement Externe*. L'ingénieur choisit cette stratégie dans le but d'ajouter un nouveau concept - *Evénement Interne* - qui peut également être une spécialisation de *Evénement*. Comme aucun élément n'est rendu caduc par l'insertion de ce nouvel élément, l'[Elément D] correspond à l'ensemble vide.

Le troisième exemple illustre une partie de méthode représentée par le concept de *Contrainte*. L'application de la stratégie SPECIALISATION permet de lui affecter la spécialisation *Contrainte d'unicité*. Comme dans le cas précédent, il n'y a pas d'élément à supprimer de la méthode.

17.2.2 GENERALISATION

Un élément de **PRODUIT** peut également être intégré dans la méthode d'origine par un lien de généralisation. La stratégie GENERALISATION permet d'effectuer des extensions de ce type.

17.2.2.1 Principe d'application de la stratégie GENERALISATION

L'élément de **PRODUIT** peut être intégré comme une spécialisation alternative à un élément de **PRODUIT** existant. C'est à dire qu'un élément du modèle existant peut se généraliser de telle façon que l'élément que l'on souhaite intégrer peut également se généraliser de la même manière.

Dans la pratique, cette stratégie insérera deux nouveaux éléments ([Élément W] et [Élément Y]). Elle créera également les deux liens de spécialisation, entre le concept généralisé ([Élément W]) et l'élément existant ([élément X]) d'une part, puis entre le concept généralisé ([élément W]) et le nouvel élément ([Élément Y]) d'autre part. Il est ensuite également nécessaire d'insérer le groupe de spécialisations correspondant aux éléments X et Y. Cependant, il est souvent nécessaire de renommer l'élément que l'on étend pour lui donner une connotation plus spécifique. La recherche d'une structure générique permettant d'effectuer ce traitement nous a conduit à l'élaboration de la séquence d'opérateurs suivante *{**Renommer** ([Élément X], [Nom X]) ; **Insérer** ([Élément W]) ; **Insérer-isa** ([Élément X], [Élément W]) ; **Insérer** [Élément Y] ; **insérer-isa** ([Élément Y], [Élément W]) ; **Insérer-Cluster** ([Elément X],[Elément Y]) }* où *[Élément Y]* représente le concept spécialisé à intégrer, *[Élément X]* le concept similaire déjà présent (dont on a modifié le nom avec le terme *[Nom X]*) et *[Élément W]* le nouveau concept généralisé.

Certains aspects de la définition de l'[Elément X] se doivent d'être généralisés également, c'est ce que permet de réaliser l'opérateur ***Généraliser*** *([Définition], [Elément X], [Elément W])*, où le terme *[Définition]* représente la partie de la définition de l'[Elément X] que l'on souhaite généraliser à l'[Elément W].

Il peut arriver qu'un concept préalablement présent dans la méthode remplisse plus ou moins les fonctions du concept que l'on veut intégrer. Cependant, si ce concept ne satisfait pas les besoins de l'ingénieur de méthodes autant que peut le faire le nouveau concept, il est nécessaire de le supprimer pour permettre un bon remplacement. Cette transformation est permise par l'opérateur de suppression ***Supprimer*** *[Élément D]*.

La figure suivante résume le principe d'application de l'opérateur GENERALISATION qui permet donc d'intégrer l'élément Y comme une alternative possible de l'élément X déjà présent dans la méthode d'origine en les généralisant avec l'élément W.

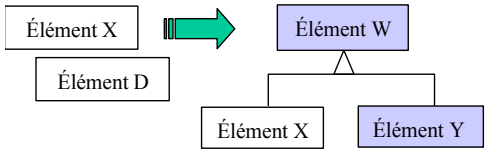
GENERALISATION ([Elément X], [Elément Y], [Elément D], [Nom X])	
	**Supprimer** [Élément D] **Renommer** ([Elément X], [Nom X]) **Insérer** [Élément W] **Insérer-isa** ([Élément X], [Élément W]) **Insérer** [Élément Y] **Insérer-isa** ([Élément Y], [Élément W]) **Insérer-Cluster** ([Elément X],[Elément Y]) **Généraliser** ([Définition], [Elément X], [Elément W])

Figure 97: Principe d'application de la stratégie GENERALISATION

17.2.2.2 Exemples d'application de la stratégie GENERALISATION

La figure suivante illustre plusieurs exemples d'application de cette stratégie.

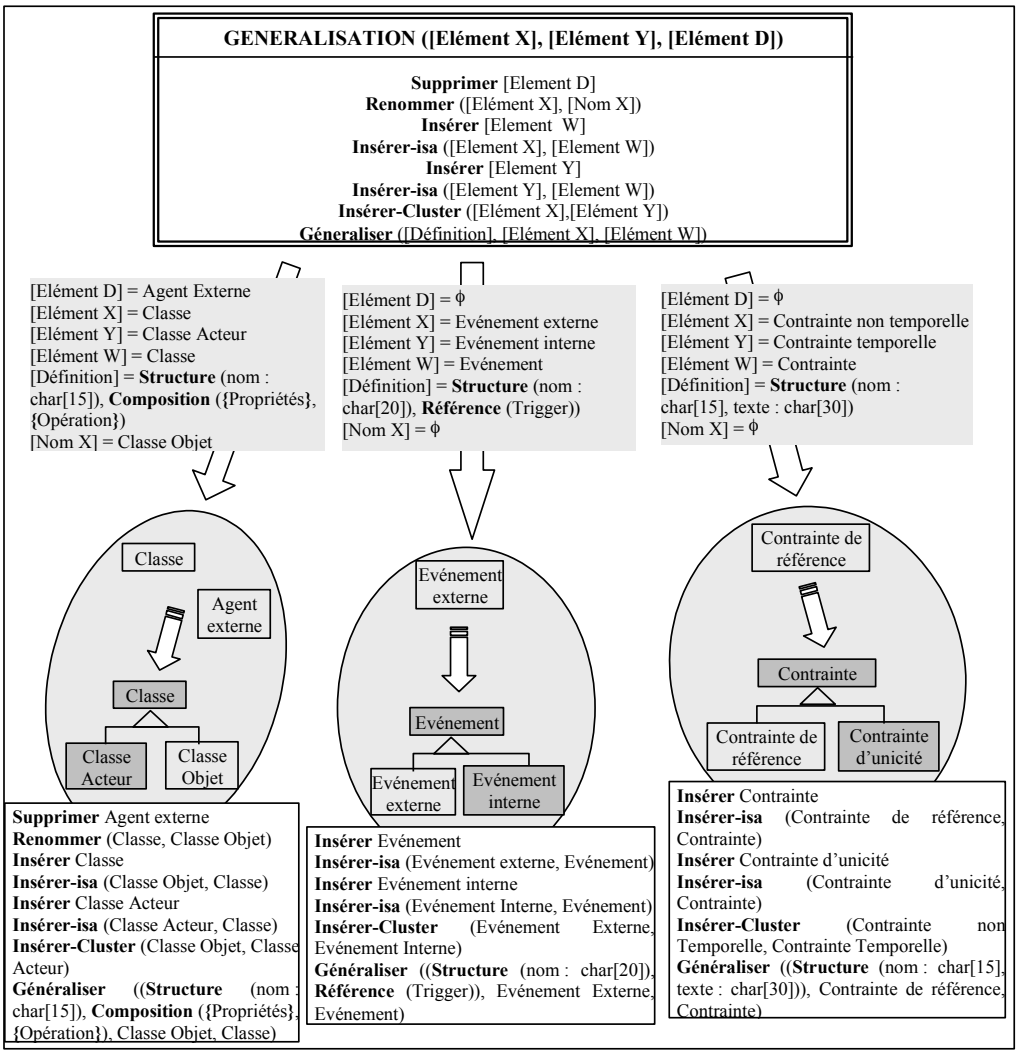


Figure 98: Exemples d'application de la stratégie GENERALISATION

Le premier exemple illustre l'application de la stratégie dans le but d'intégrer le concept de *Classe Acteur* dans une méthode contenant déjà les concepts de *Classe* et *Agent Externe*. Les modifications entraînent un remontage de *Classe* en *Classe Objet* puis une généralisation du concept de *Classe Objet* en *Classe* pour pouvoir lui affecter une autre spécialisation avec le concept de *Classe Acteur*. Comme le concept d'*Agent Externe* devient inutile du fait de l'insertion de cette nouvelle classe, cet élément est supprimé de la méthode d'origine.

Dans la méthode d'origine du second exemple est présent le concept d'*Evénement Externe*. L'application de cette stratégie permet d'insérer deux nouveaux éléments : le concept d'*Evénement* qui permettra de généraliser celui d'*Evénement Externe*, ainsi que le concept d'*Evénement Interne* qui sera une autre spécialisation d'*Evénement*.

Le troisième exemple prend en compte une méthode possédant le concept de *Contrainte de référence*. L'application de la stratégie permet de généraliser ce concept en *Contrainte* et de lui ajouter une autre spécialisation alternative, le concept de *Contrainte d'unicité*.

17.2.3 COMPOSITION

Il est également possible d'attacher un nouveau concept à la méthode d'origine par un lien de composition, ce qui est représenté par l'application de la stratégie COMPOSITION.

17.2.3.1 Principe d'application de la stratégie COMPOSITION

L'élément de **PRODUIT** est ajouté à la définition d'un élément de **PRODUIT** comme élément composant. Plus explicitement, cette stratégie insérera le nouvel élément ([Élément Y]) et créera le lien de composition entre l'élément composé ([Élément X]) et l'élément composant ([Élément Y]). L'appel de cette stratégie, contrairement à celles vues précédemment, nécessite un argument supplémentaire pour représenter la cardinalité associée au lien, cet argument est représenté sous le terme *[cardinalité]*.

La recherche d'une structure générique permettant d'effectuer ce traitement nous a conduit à l'élaboration des opérateurs *{**Insérer** [Élément Y], **Insérer-comp** ([Élément X], [Elément Y], [cardinalité])}* où *[Élément Y]* représente le nouveau concept que l'on veut intégrer, *[Élément X]* le concept déjà présent dans la méthode et *[cardinalité]* la cardinalité qui existe sur le lien de composition.

Comme pour les stratégies SPECIALISATION et GENERALISATION, il est possible que l'application de la stratégie COMPOSITION ait comme conséquence le fait qu'un élément de la méthode devienne périmé. Dans ce cas, l'opérateur de suppression ***Supprimer*** *[Élément D]* permettra d'éliminer cet élément.

On peut voir sur la figure suivante que l'élément Y est intégré comme élément composant de l'élément X existant préalablement dans la méthode d'origine.

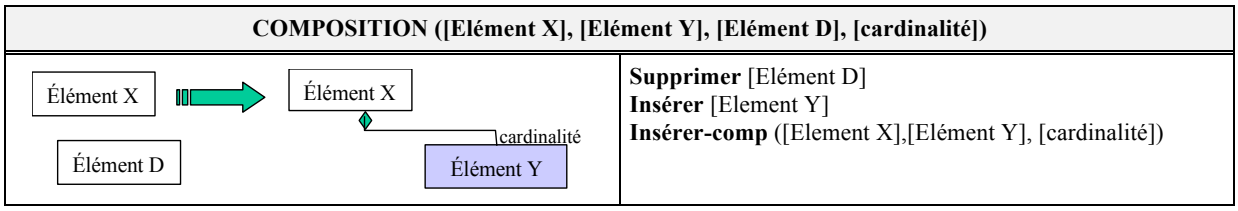


Figure 99: Principe d'application de la stratégie COMPOSITION

17.2.3.2 Exemples d'application de la stratégie COMPOSITION

La Figure 100 illustre l'application de cette stratégie sur trois exemples d'extension.

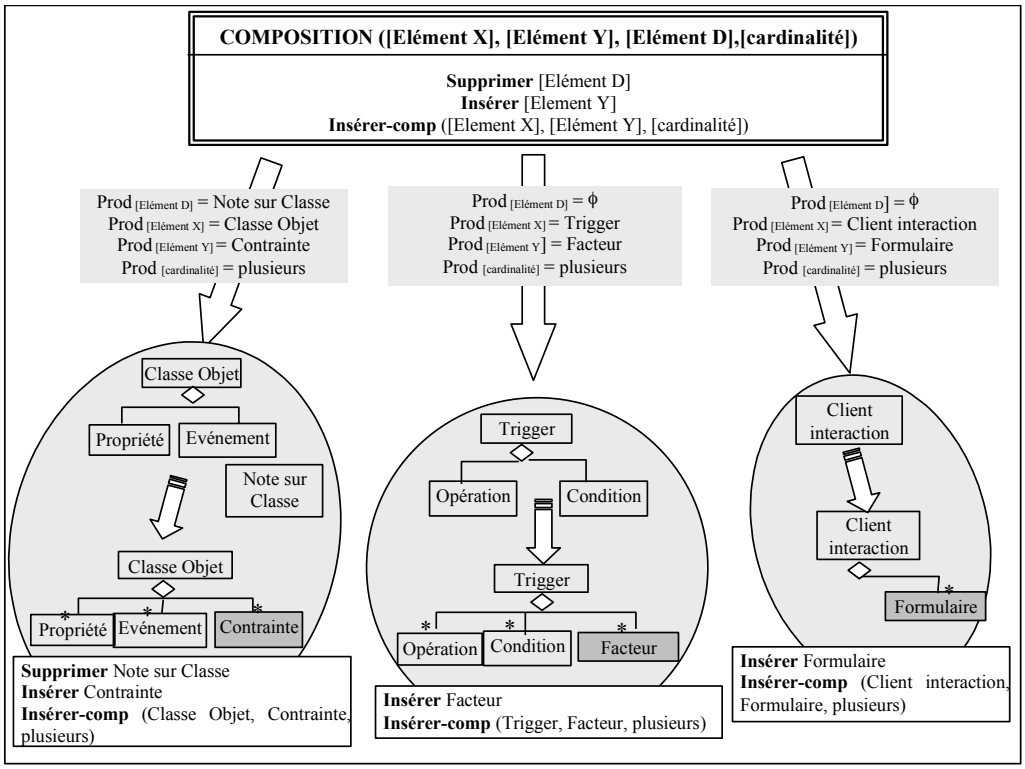


Figure 100: Exemples d'application de la stratégie COMPOSITION

Le premier exemple illustre ces opérateurs de modification en prenant pour méthode d'origine une méthode contenant le concept de *Classe Objet* composé des deux concepts *Propriété* et *Evénement*. Les contraintes attachées à une *Classe* sont ici représentées par le concept de *Note sur Classe*, simple chaîne de caractères. L'ingénieur étend cette méthode en y intégrant le nouveau concept de *Contrainte* permettant de spécifier plus strictement la définition des instances de la Classe. Ce concept est inséré comme concept composant de celui de *Classe Objet* et le concept *Note sur Classe* est supprimé de la méthode puisqu'on lui a intégré un concept équivalent plus formel et plus complet.

Dans la partie de méthode présentée dans le deuxième exemple, le concept de *Trigger* est défini comme un composé de deux autres concepts : *Opération* et *Condition*. L'ingénieur applique cette stratégie pour intégrer un autre concept dans cette définition, celui de *Facteur*.

Le troisième exemple illustre le concept de *Client Interaction* que l'ingénieur de méthodes compose avec celui de *Formulaire* en appliquant cette stratégie.

17.2.4 DECOMPOSITION

Cette stratégie permet également d'attacher un concept à la méthode d'origine par un lien de composition mais, contrairement à la stratégie COMPOSITION, la stratégie DECOMPOSITION intègre un élément comme composé de l'élément existant (et non pas comme composant).

17.2.4.1 Principe d'application de la stratégie DECOMPOSITION

Cette stratégie permet d'insérer un nouvel élément ([Elément Y]) ainsi que le lien de composition qui le relie à un concept ([Elément X]) déjà existant dans le modèle. De même que pour la stratégie précédente (COMPOSITION), l'appel de cette stratégie nécessite un argument supplémentaire pour représenter la cardinalité inhérente au lien de composition que l'on intègre entre ces deux éléments. Cet argument est définit sous le terme *[cardinalité]*.

Ces traitements peuvent s'effectuer grâce aux opérateurs génériques suivants *{**Insérer** [Élément Y], **Insérer-comp** ([Elément Y], [Élément X], [cardinalité])}* où *[Élément X]* représente le concept présent préalablement dans le modèle et *[Élément Y]* le concept à intégrer.

Remarquons que, parfois, un élément de **PRODUIT** déjà présent dans le modèle ([Élément D]) correspond plus ou moins à l'élément que l'ingénieur de méthodes souhaite intégrer. Si l'insertion de ce nouvel élément rend caduc l'utilisation de l'ancien alors il est préférable de supprimer celui-ci. L'opérateur ***Supprimer*** *[Élément D],* où *[Élément D]* représente l'élément obsolète, permet d'effectuer cette suppression.

La figure suivante visualise le principe d'application de cette stratégie et montre que l'on ajoute l'élément Y comme élément composé de l'élément X déjà présent dans la méthode d'origine.

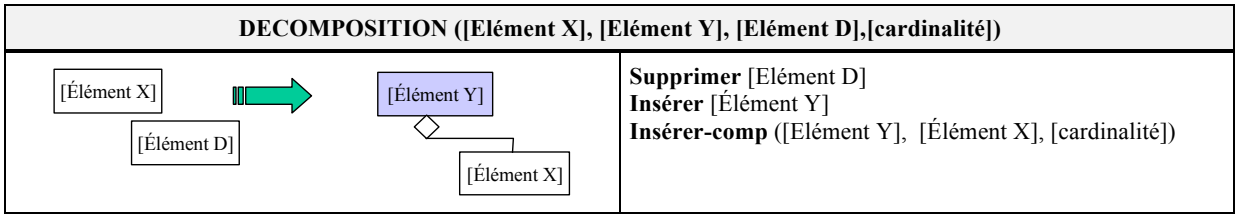


Figure 101: Principe d'application de la stratégie DECOMPOSITION

17.2.4.2 Exemples d'application de la stratégie DECOMPOSITION

La Figure 102 illustre les modifications apportées aux modèles de **PRODUIT** de deux exemples différents.

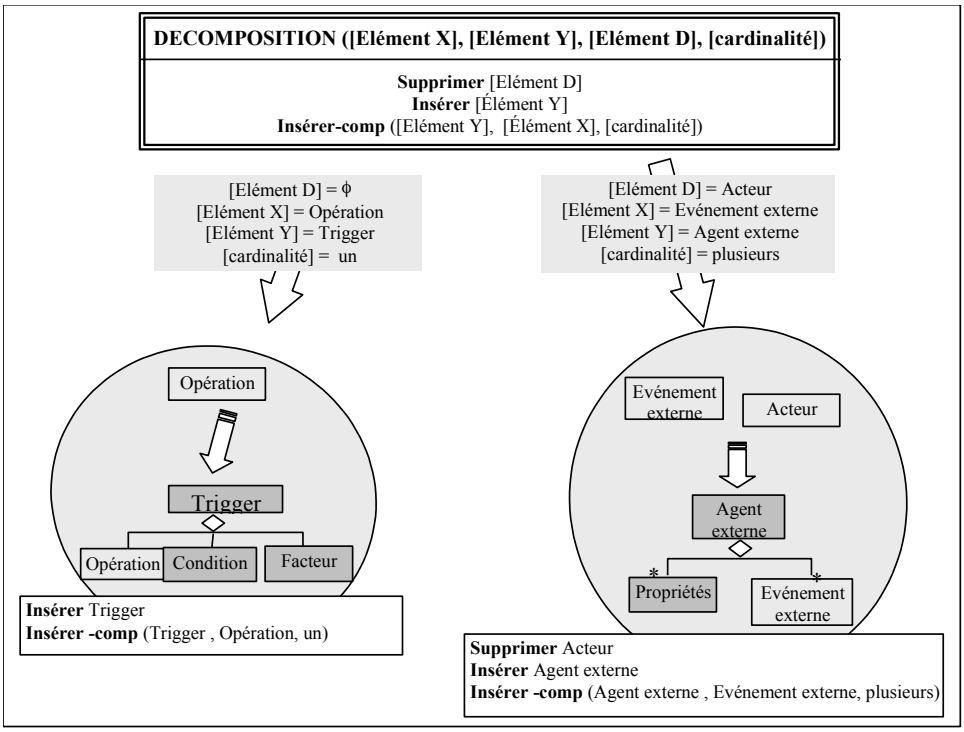


Figure 102: Exemples d'application de la stratégie DECOMPOSITION

La partie de la méthode prise en compte dans le premier exemple contient le concept d'*Opération*. Une application possible de cette stratégie sera d'intégrer le concept de *Trigger* dans la méthode comme composé de *Opération*. L'intégration de *Trigger* permet également d'intégrer toute sa définition, dont les concepts composants *Condition* et *Facteur.*

Le deuxième exemple illustre une méthode possédant à la fois le concept d'*Evénement Externe* et celui d'*Acteur*. L'ingénieur décide d'appliquer une extension de la méthode pour intégrer le concept d'*Agent Externe* qui répond mieux à ses besoins que celui d'*Acteur* présent dans la méthode. L'application de cette stratégie permet d'intégrer ce concept comme composé de celui d'*Evénement Externe*. L'insertion de ce nouvel élément se couple également avec la suppression du concept d'*Acteur*, devenu inutile avec cette modification de la méthode.

17.2.5 REFERENCE

Le dernier lien possible pour rattacher un nouvel élément au modèle de **PRODUIT** est le lien de référence. La stratégie REFERENCE permet de lier le nouvel élément à un élément déjà existant dans la méthode par ce lien de référence.

17.2.5.1 Principe d'application de la stratégie REFERENCE

Cette stratégie intègre un nouvel élément ([Élément Y]) ainsi qu'un lien de référence entre celui-ci et un concept déjà existant dans la méthode d'origine ([Élément X]). L'insertion d'un concept par le biais d'un lien de référence nécessite plusieurs arguments supplémentaires permettant de décrire les noms des rôles associés à cette référence, ainsi que les cardinalités associées. Ces arguments sont utilisés sous les termes *[cardX], [cardY],[rôleX],[rôleY]*.

La recherche d'une structure générique permettant d'effectuer ce traitement nous a conduit à l'élaboration de la séquence d'opérateurs suivante *{**Insérer** [Élément Y] ; **Insert-ref** ([Élément X], [Élément Y], [cardX], [roleX], [cardY], [roleY])}* où *[Élément Y]* représente le concept à intégrer, *[Élément X]* le concept déjà présent, *[cardX]* et *[cardY]* les cardinalités de l'association et *[roleX]* et *[roleY]* les noms des rôles attachés à cette association.

Il arrive que l'élément de **PRODUIT** que l'on intègre puisse remplacer un élément présent dans le modèle de la méthode d'origine. C'est à dire que cet élément d'origine ne correspond pas à ce que l'ingénieur de méthodes voudrait utiliser et qu'il décide de le remplacer complètement par ce nouvel élément que l'on intègre par Référence. Cet opérateur se formalise de la façon suivante : ***Supprimer** [Élément D]* où *[Élément D]* représente l'élément de la méthode d'origine que l'on souhaite remplacer.

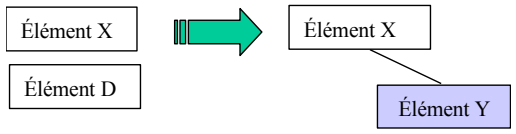
REFERENCE ([Elément X], [Elément Y], [Elément D], [cardX], [cardY], [rôleX], [rôleY])	
	**Supprimer** [Element D] **Insérer** [Element Y] **Insérer-ref** ([Elément X], [Element Y], [cardX], [roleX], [cardY], [roleY])

Figure 103: Principe d'application de la stratégie REFERENCE

Le principe d'application de cette stratégie montre bien que l'on crée une nouvelle association dans le modèle pour relier le nouvel élément Y à l'élément X de la méthode d'origine.

17.2.5.2 Exemples d'application de la stratégie REFERENCE

La Figure 104 illustre l'application de cette stratégie sur deux exemples d'extension de méthode.

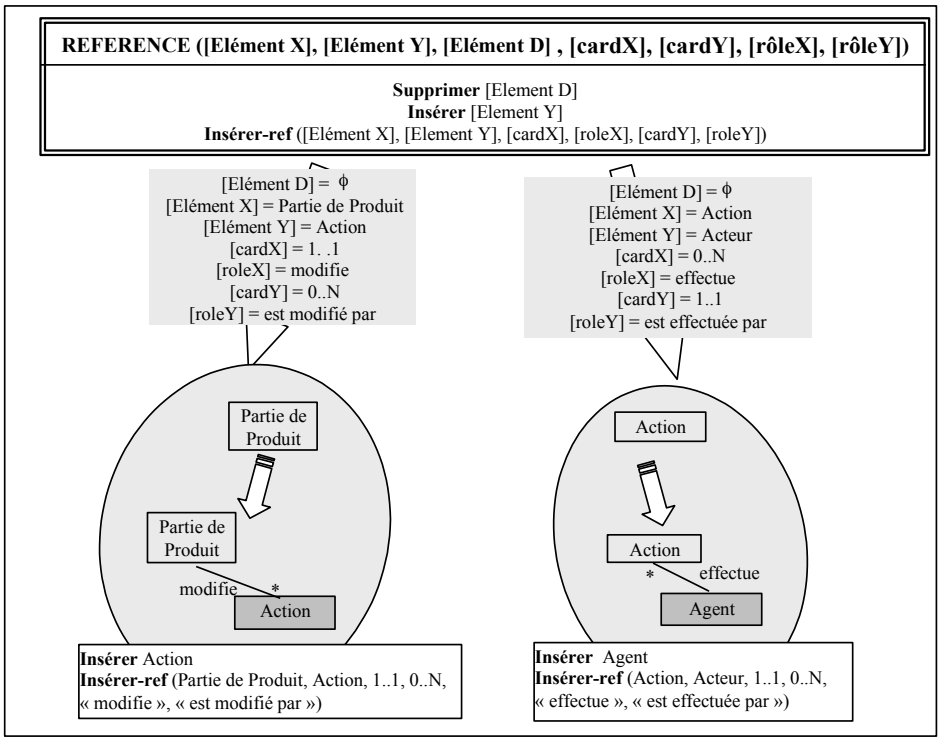


Figure 104: Exemples d'application de la stratégie REFERENCE

Le premier exemple illustre l'application de cette stratégie sur le méta-modèle d'une méthode d'analyse contenant le concept de *Partie de Produit* et à laquelle l'ingénieur de méthodes souhaite intégrer le concept d'*Action*. L'application de la stratégie REFERENCE permet d'intégrer ce concept comme concept associé à celui de *Partie de Produit*, ce qui permet d'exprimer le fait qu'une *Action* modifie une *Partie de Produit* et qu'une *Partie de Produit* peut être modifiée par une ou plusieurs *Action(s)*.

De la même façon, le deuxième exemple visualise une application possible de cette stratégie en intégrant, dans une méthode possédant le concept d'*Action*, le nouveau concept d'*Agent*. L'insertion simultanée des cardinalités et des noms de rôles permet de spécifier qu'un *Agent* peut effectuer une ou plusieurs *Action(s)* alors qu'une *Action* n'est effectuée que par un et un seul *Agent*.

17.2.6 REMPLACEMENT

Comme il a été dit dans la section 17.2, il est également possible, non pas de relier le nouvel élément au reste du modèle de la méthode d'origine, mais de remplacer un ancien élément par l'élément que l'on souhaite intégrer.

17.2.6.1 Principe d'application de la stratégie REMPLACEMENT

Cette stratégie d'extension s'applique sur les méthodes possédant un concept que l'ingénieur de méthodes trouve inadapté. Ce concept ([Élément X]) peut donc être remplacé par un autre ([Élément Y]) dans le modèle de la méthode. Cependant, les liens que l'ancien concept possède avec le reste du modèle peuvent, eux, être adaptés aux besoins de l'ingénieur. Dans ce cas, il faut pouvoir

remplacer le concept tout en conservant les liens d'héritage, de composition ou de référence qui étaient inhérents à l'ancien concept.

La recherche d'une structure générique nous a conduit à l'élaboration de l'opérateur suivant ***Remplacer*** *([Élément X],[Élément Y])* où *[Élément X]* représente l'ancien concept que l'on veut remplacer et *[Élément Y]* le nouveau concept remplaçant l'ancien.

La stratégie REMPLACEMENT est la seule stratégie d'extension de **PRODUIT** qui ne peut pas rendre un autre élément de **PRODUIT** inutile. En effet, si l'on applique cette stratégie, c'est que l'on élimine, se faisant, un élément trop semblable à celui que l'on souhaite intégrer. L'[Elément D] pour cette stratégie correspond donc toujours à l'ensemble vide et ne nécessite donc pas d'être présent dans la structure de l'opérateur.

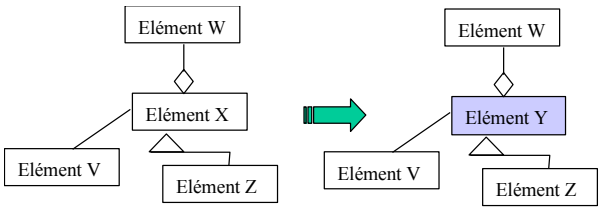
REMPLACEMENT ([Elément X], [Elément Y])	
	**Remplacer** ([Elément X], [Élément Y])

Figure 105: Principe d'application de la stratégie REMPLACEMENT

La figure précédente visualise le fait que cette technique permet d'effectuer une extension par l'application de la stratégie REMPLACEMENT. L'élément X présent dans le modèle de **PRODUIT** de la méthode d'origine est remplacé par le nouvel élément Y. Cependant, les liens que cet élément avait avec le reste du modèle – liens de composition (élément W), de référence (élément V) et de spécialisation (élément Z) - sont identiques.

17.2.6.2 Exemples d'application de la stratégie REMPLACEMENT

Cette modification est illustrée dans la Figure 106 avec trois exemples d'application de cette stratégie.

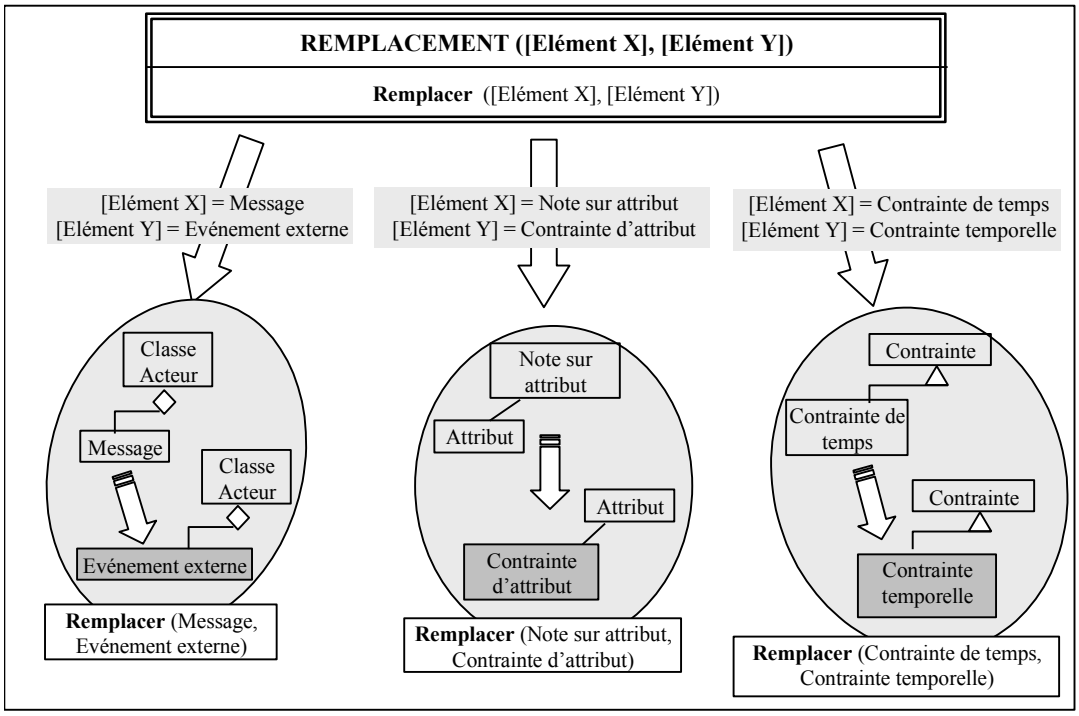


Figure 106: Exemples d'application de la stratégie REMPLACEMENT

Le premier exemple illustre le remplacement du concept de *Message* inclus dans une méthode par le concept d'*Evénement Externe*. L'application de cette stratégie conserve les liens que l'ancien concept a avec le reste du modèle, ici c'est le lien de composition avec le concept de *Classe Acteur.*

De la même manière, le deuxième exemple visualise une partie d'une méthode contenant le concept d'*Attribut* contraint par celui de *Note sur Attribut*. L'ingénieur de méthodes applique cette stratégie dans le but de remplacer ce dernier concept par celui de *Contrainte d'attribut*, plus formel. Le lien de référence déjà présent entre *Attribut* et *Note sur Attribut* reste entre *Attribut* et *Contrainte d'attribut.*

L'application de la stratégie dans le troisième exemple remplace le concept de *Contrainte de Temps* par le concept de *Contrainte Temporelle*. Le lien de spécialisation entre *Contrainte* et *Contrainte de temps* est conservé et permet désormais de spécialiser le concept de *Contrainte* en *Contrainte Temporelle*.

18 Spécification d'un Méta-Patron d'extension de Démarche

Lorsque l'ingénieur de méthodes applique un patron d'extension de **PRODUIT**, si la méthode d'origine qu'il utilise possède une partie **DÉMARCHE**, il sera préférable d'étendre également cette partie pour améliorer la complétude de la méthode. De la même façon que nous avons défini des *méta-patrons d'extension de Produit*, il est possible de définir des *méta-patrons d'extension de Démarche*.

18.1 Interface d'un méta-patron d'extension de la Démarche

Un méta-patron d'extension de la **DÉMARCHE** d'une méthode peut être décrit grâce à son interface, ainsi que l'illustre la Figure 107.

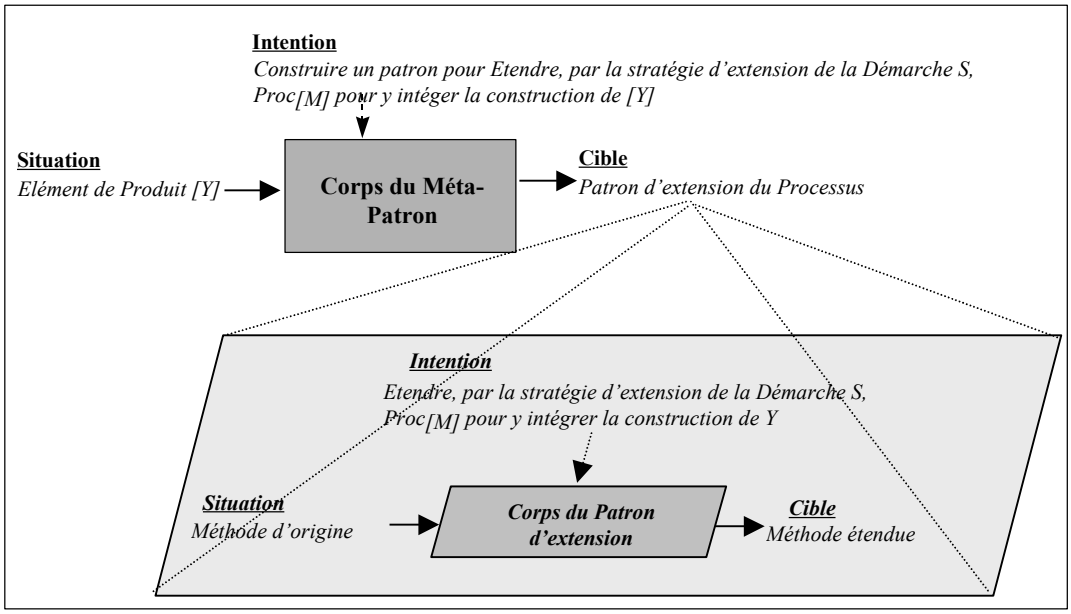


Figure 107: Interface d'un méta-patron de processus

La situation d'un méta-patron d'extension de **DÉMARCHE** correspond à la **DÉMARCHE** de construction de l'élément de **PRODUIT** dont on veut agrémenter la méthode d'origine.

L'intention d'un méta-patron d'extension de **DÉMARCHE** correspond au fait de construire un patron pour étendre la **DÉMARCHE** de la méthode d'origine pour y intégrer la construction de l'élément de **PRODUIT** en appliquant une certaine stratégie.

La cible d'un méta-patron d'extension de **DÉMARCHE** est un patron d'extension de la **DÉMARCHE** pour la construction d'un élément de **PRODUIT** défini et suivant une certaine stratégie d'extension.

Le corps d'un méta-patron correspond donc à une séquence d'opérateurs permettant de construire ce patron d'extension: sa situation, son intention, sa cible et son corps. En ce qui concerne cette dernière partie, chaque stratégie utilisée représente une suite d'opérateurs de modification permettant l'extension d'une méthode. Les stratégies d'extension de la partie **DÉMARCHE** d'une méthode sont décrites dans la section suivante.

18.2 Stratégies d'extension de la DÉMARCHE d'une méthode

Une extension de la **DÉMARCHE** d'une méthode revient à greffer de nouvelles démarches dans l'arbre de processus de la méthode existante pour permettre la prise en compte de la construction des nouveaux concepts. Deux possibilités sont offertes à l'ingénieur de méthodes à ce niveau : soit l'étape de construction du schéma objet de la méthode d'origine est laissée intacte et les greffes des nouveaux

processus sont intégrées de façon contrôlée, soit l'arbre de processus est modifié pour intégrer complètement les nouveaux processus de façon transparente pour l'utilisateur.

Intégration contrôlée

Dans une extension de ce type, les processus permettant d'étendre la méthode sont greffés de façon à laisser la démarche de construction du schéma OO inchangée. En fait, l'ingénieur de méthodes ne commence à étendre les éléments du schéma qu'à partir du moment où ils ont été entièrement décrits de façon non étendue. L'arbre de la démarche de construction de la méthode d'origine conserve donc ses processus de construction (« Construire X ») mais il est incrémenté de nouveaux processus d'extension (« Etendre X »). Ces deux processus se déroulant en séquence, on appelle cette technique une extension Séquentielle.

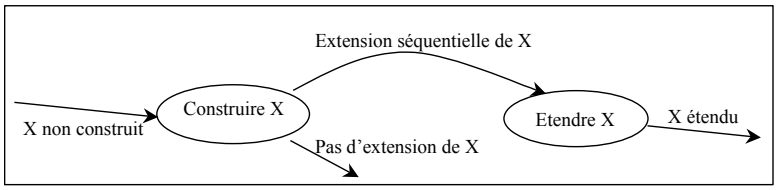


Figure 108: Principe de l'extension séquentielle

Le principe de l'extension illustré par la Figure 108 visualise la séquence des deux processus de la démarche d'extension séquentielle qui sont la construction et l'extension de X. Cependant, le terme *X* peut avoir un sens différent selon la granularité qui intéresse l'ingénieur de méthodes. En effet, à la granularité la plus basse, X représente un élément simple de la méthode à étendre ; c.-à-d. la construction du concept de *Classe Objet* ou encore de celui de *Granule*, etc. A l'inverse, à la granularité la plus haute, X correspond au schéma entier de la méthode. Ces deux cas sont illustrés par la Figure 109 suivante.

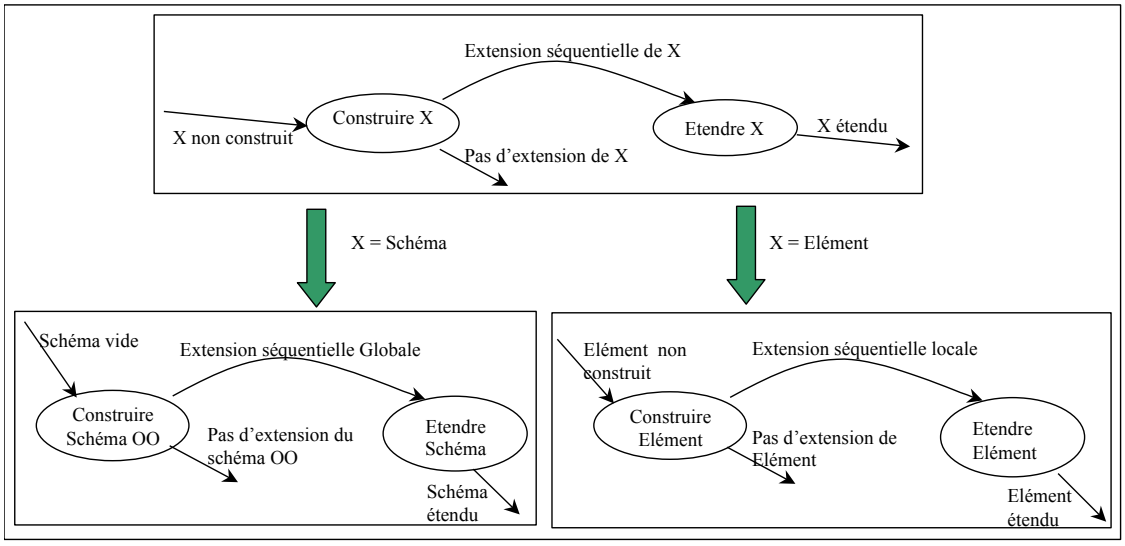


Figure 109: Principe de l'extension séquentielle

Dans le premier cas, une extension séquentielle représentera une séquence de l'arbre de processus qui permettra d'enchaîner la construction et l'extension d'un élément simple, alors que dans le deuxième

cas, l'arbre de processus mettra en séquence la construction entière du schéma objet avec son extension. Ainsi que l'indique la figure précédente, ces deux possibilités sont des stratégies appelées respectivement EXTENSION SEQUENTIELLE LOCALE et EXTENSION SEQUENTIELLE GLOBALE.

Le premier exemple de la Figure 110 illustre une extension d'une méthode de façon séquentielle globale pour y intégrer le processus de construction du concept de *Classe Calendrier* alors que le deuxième exemple illustre une extension de façon séquentielle locale pour intégrer le processus de construction du concept de *Contrainte* dans la méthode.

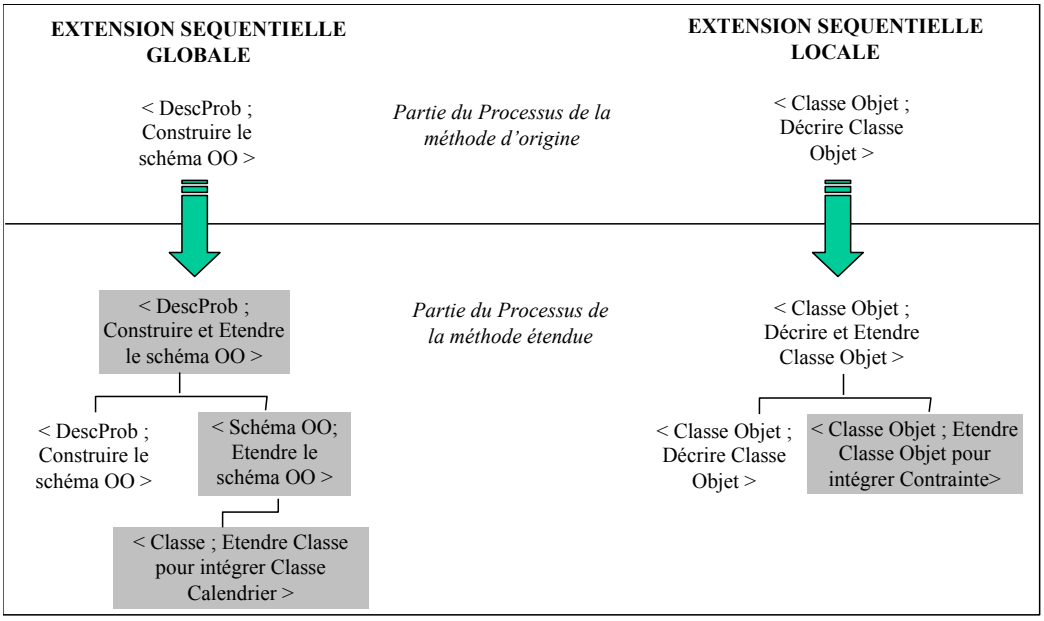


Figure 110: Exemples d'applications des stratégies d'EXTENSION SEQUENTIELLE

Intégration transparente

Contrairement aux extensions séquentielles qui permettent de conserver intact les processus de constructions d'origine, ce type d'extension intègre de façon transparente les modifications apportées à l'arbre de processus. L'extension n'apparaît donc plus dans le processus comme une étape artificielle mais comme une étape plus complète. En conséquence, l'ingénieur de méthodes ne doit plus progresser dans une séquence « construction, extension » mais exécute simplement un arbre de processus prenant les modifications en compte dès le départ de l'exécution. Ce principe est illustré par la Figure 111 suivante.

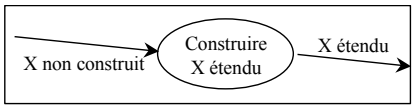


Figure 111: Principe de l'extension intégrée

Ce type d'extension est représenté par la stratégie appelée EXTENSION INTEGREE.

L'exemple de la Figure 112 illustre le cas d'une méthode que l'on étend de façon intégrée pour y ajouter le processus de construction du concept de *Facteur*.

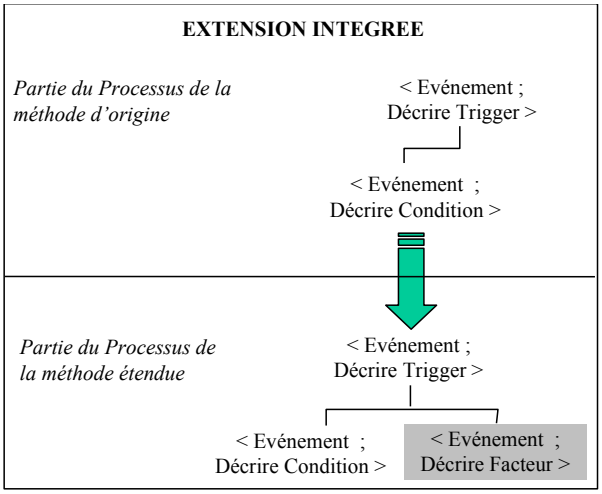


Figure 112: Exemples d'application de la stratégie EXTENSION INTEGREE

La **DÉMARCHE** d'une méthode peut donc être étendue de trois façons différentes, représentant les trois types de stratégies différentes d'extension de la **DÉMARCHE**.

18.3 Stratégie d'extension de PROCESSUS *Versus* Stratégie d'extension de PRODUIT

Il est à noter que l'application de telle ou telle stratégie d'extension de **PRODUIT** se répercute sur l'utilisation de la stratégie d'extension de **DÉMARCHE**. Il y a trois types de répercutions possibles.

Nous pouvons remarquer que les stratégies d'extension du **PRODUIT** SPECIALISATION ou encore GENERALISATION entraîneront, intrinsèquement, des modifications des alternatives (*choix*) possibles de la branche de processus concernant un élément de **PRODUIT**. Ceci s'effectue par une augmentation des possibilités de types spécialisés (lors d'une Spécialisation) ou d'une étape supplémentaire d'imbrication (en cas de Généralisation).

De la même façon, les stratégies d'extension du **PRODUIT** COMPOSITION et DECOMPOSITION, eux, entraîneront une modification de la séquence (*plan*) concernant un élément de **PRODUIT**.

En ce qui concerne la stratégie d'extension du **PRODUIT** REMPLACEMENT, il s'agira également d'un remplacement mais au niveau de l'arbre de processus, c'est à dire que l'on va greffer une nouvelle branche de processus à la place d'une branche existante.

On peut également différencier deux cas possibles au niveau d'une EXTENSION SEQUENTIELLE GLOBALE. En effet, soit l'extension que l'ingénieur de méthodes veut greffer est la première, soit il a déjà effectué une **DÉMARCHE** d'extension du processus. Dans ce dernier cas, il faut réévaluer le graphe de précédence correspondant au plan de l'extension pour permettre à la méthode de conserver sa cohérence et son intégrité. Nous distinguerons ces deux cas par les termes de « 1ère Greffe » et de « Greffe Additionnelle ».

Il existe une autre différentiation à prendre en compte lors des stratégies d'extension par SPECIALISATION et par GENERALISATION. En effet, l'extension sera différente selon la spécialisation spécifique de l'élément considéré, c.-à-d. si il a une spécialisation par type ou bien par état. C'est la raison pour laquelle une autre classification que l'on appellera « par type » ou « par état » a été introduite.

Nous avons défini plusieurs stratégies d'extension de la **DÉMARCHE** différentes pouvant s'exécuter après chacune des stratégies d'extension de **PRODUIT** utilisé. Il n'est toutefois pas toujours possible de les exécuter quelle que soit la stratégie d'extension de **PRODUIT** choisi. De plus, selon la stratégie d'extension de **PRODUIT** utilisée, les modifications de la **DÉMARCHE** ne seront pas les mêmes. La Figure 113 visualise les possibilités d'exécution de ces stratégies.

Stratégies de Démarche / Stratégies de Produit	EXTENSION SEQUENTIELLE GLOBALE		EXTENSION SEQUENTIELLE LOCALE	EXTENSION INTEGREE	
	1ère greffe	Greffe Additionnelle		Élément ayant une spécialisation par type	Élément ayant une spécialisation par état
SPECIALISATION	SPECIALISATION GLOBALE (1ere Greffe)	SPECIALISATION GLOBALE (Greffe Additionnelle)	SPECIALISATION LOCALE	SPECIALISATION INTEGREE (par type)	SPECIALISATION INTEGREE (par état)
GENERALISATION	GENERALISATION GLOBALE (1ere Greffe)	GENERALISATION GLOBALE (Greffe Additionnelle)	GENERALISATION LOCALE	GENERALISATION INTEGREE (par type)	GENERALISATION INTEGREE (par état)
COMPOSITION	COMPOSITION GLOBALE (1ere Greffe)	COMPOSITION GLOBALE (Greffe Additionnelle)	COMPOSITION LOCALE	COMPOSITION INTEGREE	
DECOMPOSITION	DECOMPOSITION GLOBALE (1ere Greffe)	DECOMPOSITION GLOBALE (Greffe Additionnelle)	DECOMPOSITION LOCALE	DECOMPOSITION INTEGREE	
REFERENCE	REFERENCE GLOBALE (1ere Greffe)	REFERENCE GLOBALE (Greffe Additionnelle)	REFERENCE LOCALE	REFERENCE INTEGREE	
REMPLACEMENT				REMPLACEMENT INTEGRE	

Figure 113: Stratégies d'extension de la DÉMARCHE *Versus* Stratégies d'extension de PRODUIT

On peut remarquer dans cette figure qu'il n'est pas permis à l'ingénieur de méthodes d'exécuter les stratégies d'extension de la **DÉMARCHE** EXTENSION SEQUENTIELLE GLOBALE ou EXTENSION SEQUENTIELLE LOCALE si la stratégie d'extension de **PRODUIT** était REMPLACEMENT. En effet, il serait inutile de parcourir le processus de construction OO pour un élément s'il fallait le remplacer dès que l'on passe à l'extension. Mieux vaut, dans ce cas, n'autoriser que la stratégie EXTENSION INTEGREE pour éviter tout travail inutile.

On obtient donc en réalité 23 stratégies différentes d'extension de la **DÉMARCHE** d'une méthode que l'on peut appliquer sur une méthode orientée objets.

18.3.1 SPECIALISATION GLOBALE (1ère Greffe)

On applique cette stratégie lorsque la partie **PRODUIT** a été étendue par la stratégie SPECIALISATION et que l'ingénieur de méthodes choisit d'étendre le processus par la stratégie EXTENSION SEQUENTIELLE GLOBALE (1ère Greffe).

18.3.1.1 Principe d'application de la stratégie SPECIALISATION GLOBALE (1ère Greffe)

Le premier opérateur de modification lors de l'application d'une stratégie d'extension du **PRODUIT** est une suppression d'un possible élément de **PRODUIT** rendu inutile par l'extension de la méthode. De la même manière, il faut, lors de l'extension de la partie **DÉMARCHE**, supprimer toute trace de cet élément. Ceci se fait par la suppression de la **DÉMARCHE** de construction de cet élément dans l'arbre de processus. Cet opérateur peut s'écrire de la manière suivante : ***Supprimer*** $Proc_{[Elément\ D]}$, où *[Elément D]* représente l'élément inutile et $Proc_{[Elément\ D]}$ sa démarche de description.

Le fait qu'il n'y ait pas eu de greffe préalable suivant cette stratégie particulière signifie qu'il va falloir créer une nouvelle branche de l'arbre de processus pour prendre en compte toute nouvelle extension Séquentielle Globale. En fait, il suffit d'ajouter une nouvelle racine à l'arbre qui permettra de mettre en séquence la construction orientée objet initiale puis les modifications d'extension. La recherche d'une structure générique permettant d'effectuer ce traitement nous a conduit à l'élaboration de la séquence d'opérateurs suivante : {***Insérer nœud*** < *Description du Problème; Construire et étendre le schéma OO* > ; ***Insérer arc-plan entre*** < *Description du Problème; Construire et étendre le schéma OO* > ***et*** *Racine* ($Proc_{[M]}$) }. Ici, le nœud appelé *Racine* ($Proc_{[M]}$) correspond à la racine de l'arbre de processus qui concerne la construction du schéma Objet avant toute extension.

Greffer une nouvelle branche, comme père de la racine antérieure de l'arbre de processus, insère également toutes les démarches sous-jacentes. En effet, cette stratégie particulière insère le nœud racine mais aussi un nœud fils spécifique aux extensions (appelé <*Schéma OO; Etendre le schéma OO*>) ne contenant après l'exécution de cette stratégie qu'une seule démarche étant < *[Élément X]; Etendre, par SPECIALISATION GLOBALE (1ère Greffe), [Élément X] pour intégrer [Élément Y]* >.

La dernière étape consiste à construire le graphe de précédence de cette séquence, ce qui se fait grâce à l'opérateur ***Insérer Graphe-Précédence pour*** < *Description du Problème; Construire et étendre le schéma 00* >.

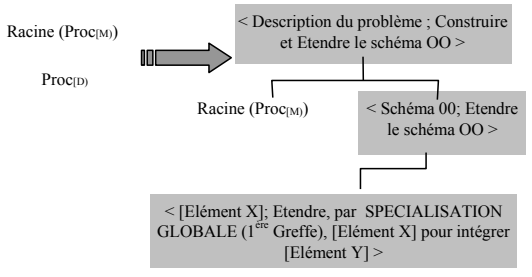
SPECIALISATION GLOBALE (1ère Greffe) ([Elément X], [Elément Y], [Elément D])	
	**Supprimer** $Proc_{[D]}$ **Insérer branche** < Description du Problème; Construire et étendre le schéma OO > **Insérer arc-plan entre** < Description du Problème; Construire et étendre le schéma OO > **et** Racine ($Proc_{[M]}$) **Insérer Graphe-Précédence pour** < Description du Problème; Construire et étendre le schéma OO >

Figure 114 : Principe d'application de la stratégie SPECIALISATION GLOBALE (1ère Greffe)

18.3.1.2 Exemples d'application de la stratégie SPECIALISATION GLOBALE (1ère Greffe)

La Figure 115 illustre l'application de cette stratégie sur deux exemples d'extension.

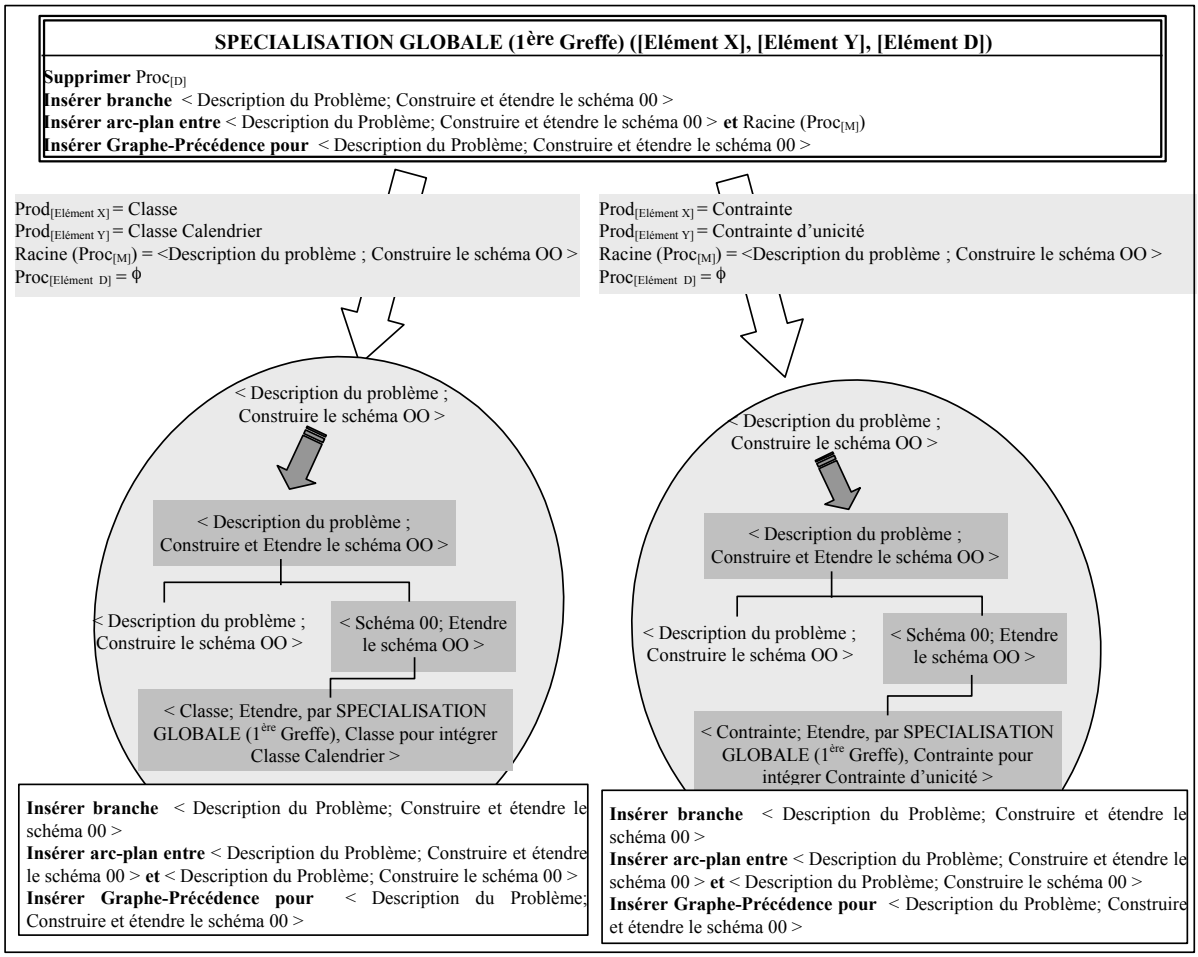


Figure 115: Exemples d'application de la stratégie SPECIALISATION GLOBALE (1ère Greffe)

Ces deux exemples illustrent des arbres de processus dont la racine représente la démarche de construction du schéma Objet. L'application de cette stratégie sur ces deux méthodes permet d'intégrer une nouvelle racine à ces arbres dans le but de pouvoir mettre en séquence la construction du schéma Objet et l'extension de ce dernier. C'est sur la branche correspondant aux extensions que l'ingénieur de méthodes greffera l'extension spécifique qui l'intéresse. Dans le premier cas, c'est la branche correspondant à la spécialisation du concept de *Classe* en *Classe Calendrier* alors que dans le second cas il s'agit de la spécialisation du concept de *Contrainte* en *Contrainte d'unicité*.

18.3.2 SPECIALISATION GLOBALE (Greffe Additionnelle)

Cette stratégie s'applique lorsque l'ingénieur de méthodes a déjà préalablement exécuté un patron d'extension de la **DÉMARCHE** avec la stratégie EXTENSION SEQUENTIELLE GLOBALE, que la partie **PRODUIT** a été étendue grâce à la stratégie SPECIALISATION et qu'il souhaite de nouveau exécuter la stratégie d'extension EXTENSION SEQUENTIELLE GLOBALE.

18.3.2.1 Principe d'application de la stratégie SPECIALISATION GLOBALE (Greffe Additionnelle)

Comme pour l'opérateur précédent, le premier opérateur de transformation d'une stratégie d'extension du **PRODUIT** étant une suppression d'un possible élément de **PRODUIT** rendu caduc par l'extension de la méthode, il faut, lors de l'extension de la partie **DÉMARCHE**, supprimer toute trace de cet élément. Il est donc nécessaire de supprimer la **DÉMARCHE** de construction de cet élément dans l'arbre de processus. Cet opérateur s'écrit de la manière suivante : ***Supprimer*** $Proc_{[Elément\ D]}$, où *[Elément D]* représente l'élément inutile et $Proc_{[Elément\ D]}$ son processus de construction.

Le fait que la partie **DÉMARCHE** ait déjà été étendue avec cette stratégie permet de savoir qu'il existe déjà une branche de l'arbre de processus qui concerne les extensions suivant cette stratégie et qu'il suffit donc d'ajouter une nouvelle démarche dans la séquence pour prendre en compte celle qui nous intéresse. Cet ajout se fait grâce à l'opérateur générique suivant ***Insérer arc-plan entre*** *Extension* ($Proc_{[M]}$) ***et*** *< [Élément X] ; Etendre, par SPECIALISATION GLOBALE (Greffe Additionnelle), [Élément X] pour intégrer [Élément Y]>* où [Élément X] représente l'élément qui était déjà présent dans la méthode d'origine, [Élément Y] l'élément qui a été intégré grâce à un patron d'extension du **PRODUIT** et *Extension*($Proc_{[M]}$) la démarche d'extension du schéma OO.

Il est ensuite nécessaire de réorganiser le graphe de transition d'états de la séquence de ce processus car certaines extensions peuvent avoir des heuristiques de précédence entre elles. Ceci se fait avec l'opérateur ***Changer Graphe-Précédence pour*** *Extension (Proc[M]).*

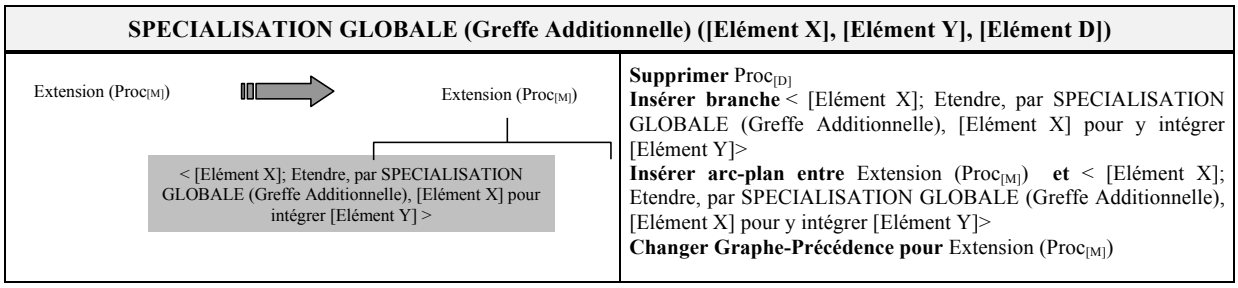


Figure 116 : Principe d'application de la stratégie SPECIALISATION GLOBALE (Greffe Additionnelle)

18.3.2.2 Exemples d'application de la stratégie SPECIALISATION GLOBALE (Greffe Additionnelle)

Deux exemples illustrent cette stratégie dans la figure suivante.

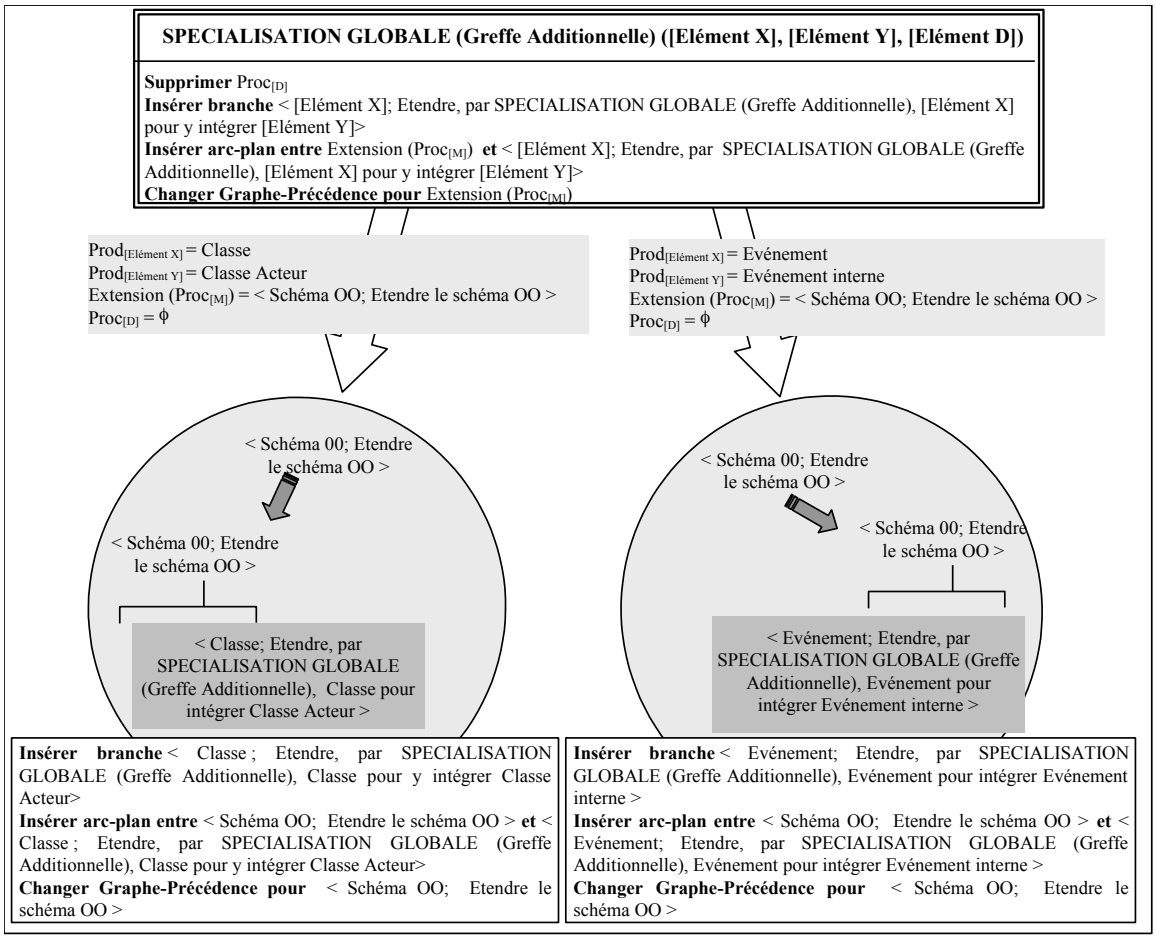


Figure 117: Exemples d'application de la stratégie SPECIALISATION GLOBALE (Greffe Additionnelle)

Ces exemples illustrent tous deux des méthodes ayant été étendues auparavant par la stratégie d'extension de **DÉMARCHE** EXTENSION GLOBALE. En effet, leurs **DÉMARCHE**s possèdent chacune un nœud représentant l'extension séquentielle globale de la méthode. Dans le premier exemple, la méthode est étendue en ajoutant la **DÉMARCHE** de description du concept de *Classe Acteur* par spécialisation de *Classe*, alors que dans le second exemple, c'est la description du concept d'*Evénement interne* qui est intégrée comme spécialisation de celle d'*Evénement*.

18.3.3 SPECIALISATION LOCALE

On applique cette stratégie lorsque la partie **PRODUIT** a été étendue par la stratégie SPECIALISATION et que l'ingénieur de méthodes choisit d'étendre la partie **DÉMARCHE** par la stratégie EXTENSION SEQUENTIELLE LOCALE.

18.3.3.1 Principe d'application de la stratégie SPECIALISATION LOCALE

Comme pour toute stratégie d'extension de **PRODUIT,** le premier opérateur de transformation est une suppression d'un possible élément de **PRODUIT** rendu inutile par l'extension de la méthode. Alors, lors de l'extension de la partie **DÉMARCHE**, il est nécessaire de supprimer également toute trace de cet élément dans l'arbre de processus (suppression de la **DÉMARCHE** de construction de cet

élément). Cet opérateur peut s'écrire de la manière suivante : ***Supprimer*** $Proc_{[Elément\ D]}$, où *[Elément D]* représente l'élément inutile et $Proc_{[Elément\ D]}$ son processus de construction.

L'application de cette stratégie permet de greffer un processus intermédiaire entre le processus de description de l'élément que l'on veut étendre - $Proc_{[Elément\ X]}$ - et son père dans l'arbre de processus - *Parent-de* ($Proc_{[Elément\ X]}$). Ce processus intermédiaire permettra de mettre en séquence la **DÉMARCHE** de description et une nouvelle **DÉMARCHE** d'extension de ce même élément. La recherche d'une structure générique permettant d'effectuer ce traitement nous a conduit à l'élaboration des opérateurs suivants.

L'opérateur ***Supprimer arc entre*** $Proc_{[Elément\ X]}$ ***et*** *Parent-de* ($Proc_{[Elément\ X]}$) permet de couper le lien entre le processus de description de l'Elément X et son processus père.

L'opérateur permettant de greffer le processus intermédiaire se formalise de la façon suivante : ***Insérer nœud*** *< [Élément X]; Décrire et étendre [Élément X] >* . Il est ensuite nécessaire de définir si l'arc entre la démarche de description de X et son père est un plan ou un choix, de manière à insérer le processus intermédiaire avec un arc du bon type. Ceci se fait avec la séquence d'opérateurs suivante :

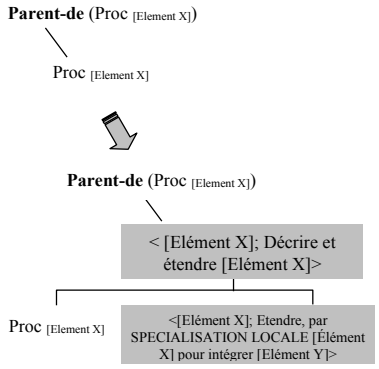
Si *Type-Parent-de* ($Proc_{[Elément\ X]}$) *= PLAN* ***alors***
Insérer arc-plan entre *< [Élément X]; Décrire et Étendre [Élément X] >* ***et***
Parent-de ($Proc_{[Elément\ X]}$)
sinon
Insérer arc-choix entre *< [Élément X]; Décrire et Étendre [Élément X] >* ***et***
Parent-de ($Proc_{[Elément\ X]}$)
fsi.

On insère ensuite l'arc de type plan permettant de relier ce processus intermédiaire et la démarche de description de l'élément X avec l'opérateur ***Insérer arc-plan entre*** *< [Élément X]; Décrire et Étendre [Élément X] >* ***et*** $Proc_{[Elément\ X]}$.

Les opérateurs *{* ***Insérer branche*** *< [Élément X]; Etendre, par SPECIALISATION LOCALE, [Élément X] pour intégrer [Élément Y] > ;* ***Insérer arc-plan entre*** *< [Élément X]; Décrire et Étendre [Élément X] >* ***et*** *< [Élément X];]; Etendre, par SPECIALISATION LOCALE, [Élément X] pour intégrer [Élément Y] >}* permettent de mettre en séquence la démarche de construction et la démarche d'extension.

La dernière étape consiste à construire le graphe de précédence de cette séquence d'extension, ce qui se fait grâce à l'opérateur ***Insérer Graphe-Précédence pour*** *< [Élément X]; Décrire et Étendre [Élément X] >*.

SPECIALISATION LOCALEMENT ([Elément X], [Elément Y], [Elément D])



Supprimer $Proc_{[Elément D]}$
Supprimer arc entre $Proc_{[Elément X]}$ **et** Parent-de ($Proc_{[Elément X]}$)
Insérer nœud < [Elément X]; Décrire et étendre [Elément X] >
Si Type-Parent-de ($Proc_{[Elément X]}$) = PLAN **alors**
Insérer arc-plan entre < [Élément X]; Décrire et Étendre [Élément X] > **et** Parent-de ($Proc_{[Elément X]}$)
sinon
Insérer arc-choix entre < [Élément X]; Décrire et Étendre [Élément X] > **et** Parent-de ($Proc_{[Elément X]}$)
fsi
Insérer arc-plan entre < [Elément X]; Décrire et Etendre [Elément X] > **et** $Proc_{[Elément X]}$
Insérer branche < [Elément X]; Etendre, par SPECIALISATION LOCALE [Elément X] pour intégrer [Elément Y] >
Insérer arc-plan entre < [Elément X]; Décrire et Etendre [Elément X] > **et** < [Elément X]; Etendre, par SPECIALISATION LOCALE [Elément X] pour intégrer [Elément Y] >
Insérer Graphe-Précédence pour < [Elément X]; Décrire et Etendre [Elément X] >

Figure 118 : Principe d'application de la stratégie SPECIALISATION LOCALE

18.3.3.2 Exemples d'application de la stratégie SPECIALISATION LOCALE

La figure suivante illustre l'application de cette stratégie sur deux exemples.

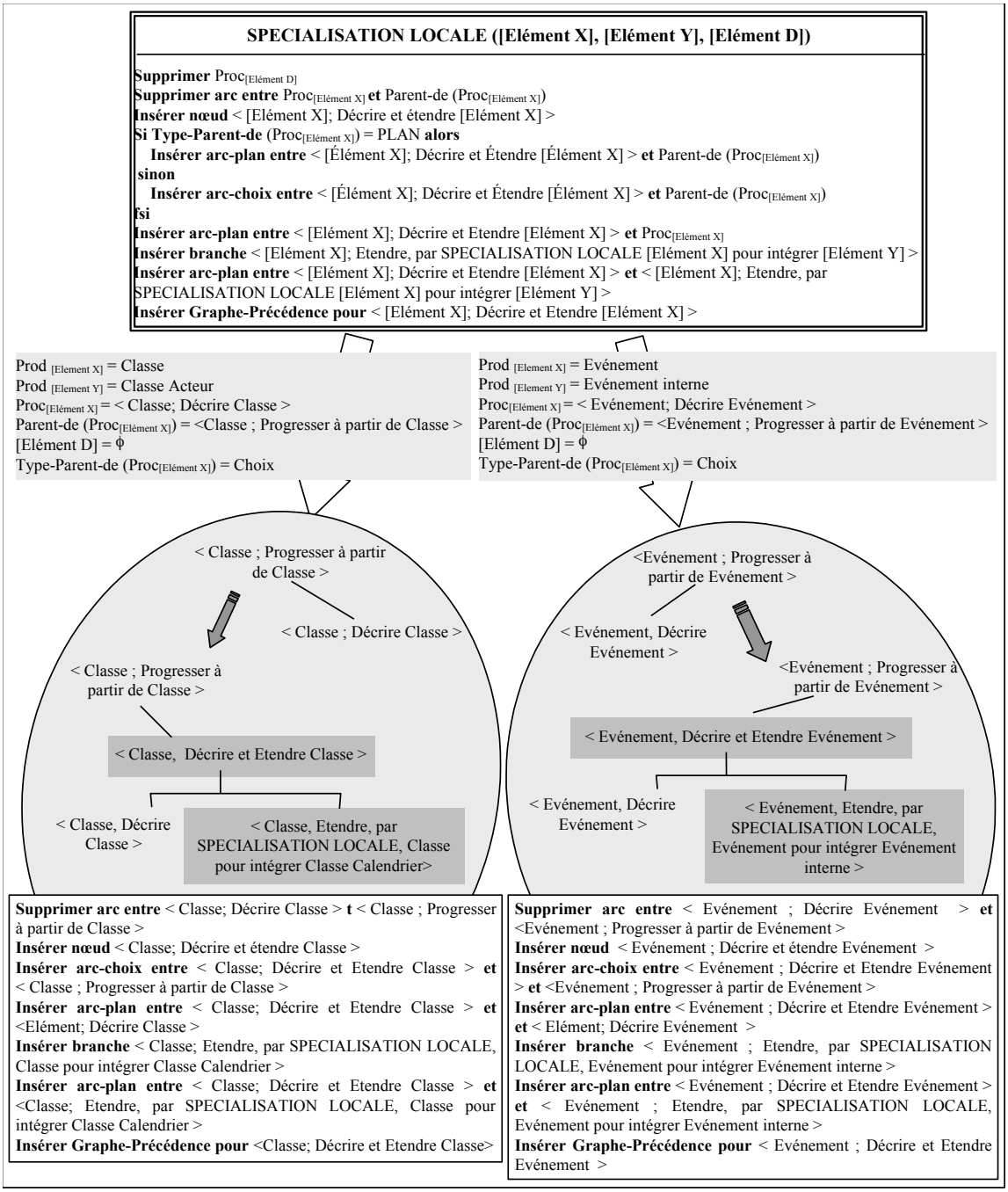


Figure 119: Exemples d'application de la stratégie SPECIALISATION LOCALE

Ces deux exemples montrent l'application de la stratégie SPECIALISATION LOCALE pour deux extensions spécifiques. La première permet d'ajouter, à la **DÉMARCHE** de construction du concept de *Classe*, la possibilité de décrire une nouvelle spécialisation de ce concept : la *Classe Calendrier*. La deuxième extension, elle, intègre la construction du concept d'*Evénement Interne* comme nouvelle alternative à la notion de concept d'*Evénement*. Ces applications permettent de parcourir l'arbre de processus en commençant par décrire le concept généralisé tout d'abord et, ensuite, de le spécialiser avec le nouvel élément.

18.3.4 SPECIALISATION INTEGREE (par type)

On applique cette stratégie lorsque la partie **PRODUIT** de la méthode d'origine a été étendue avec la stratégie d'extension de **PRODUIT** SPECIALISATION et que l'ingénieur de méthodes souhaite étendre la partie **DÉMARCHE** de la méthode avec la stratégie générique EXTENSION INTEGREE. Cette stratégie spécifique ne peut toutefois s'exécuter que si l'élément spécialisé possède une spécialisation par type. Si, au contraire, il a une spécialisation par état, la stratégie à appliquer sera celle de la section suivante.

18.3.4.1 Principe d'application de la stratégie SPECIALISATION INTEGREE (par type)

Pour la même raison que précédemment, il est utile de supprimer toute trace d'un élément rendu caduc dans l'arbre de processus (suppression de la **DÉMARCHE** de construction de cet élément). Cet opérateur peut s'écrire de la manière suivante : ***Supprimer*** $Proc_{[Elément\ D]}$, où *[Elément D]* représente l'élément inutile et $Proc_{[Elément\ D]}$ sa démarche de construction.

L'application de cette stratégie spécifique permet de greffer, à la **DÉMARCHE** de description d'un élément déjà existant, une nouvelle alternative correspondant à la **DÉMARCHE** de construction de l'élément intégré lors de l'extension de la partie **PRODUIT**. Cette transformation de l'arbre de processus s'effectue donc en ajoutant une nouvelle branche et en reliant celle-ci à l'arbre. Ceci s'écrit de la façon suivante : *{**Insérer branche** < [Élément Y]; Décrire [Elément Y] > ; **Insérer arc-choix entre** $Proc_{[Elément\ X]}$ **et** < [Élément Y]; Décrire [Elément Y] >}*.

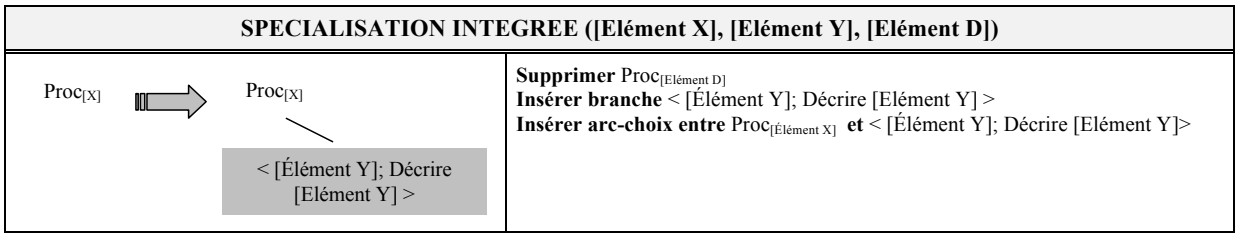


Figure 120 : Principe d'application de la startégie SPECIALISATION INTEGREE (par type)

18.3.4.2 Exemples d'application de la stratégie SPECIALISATION INTEGREE (par type)

L'application de cette stratégie est illustrée dans la figure suivante sur deux exemples représentatifs.

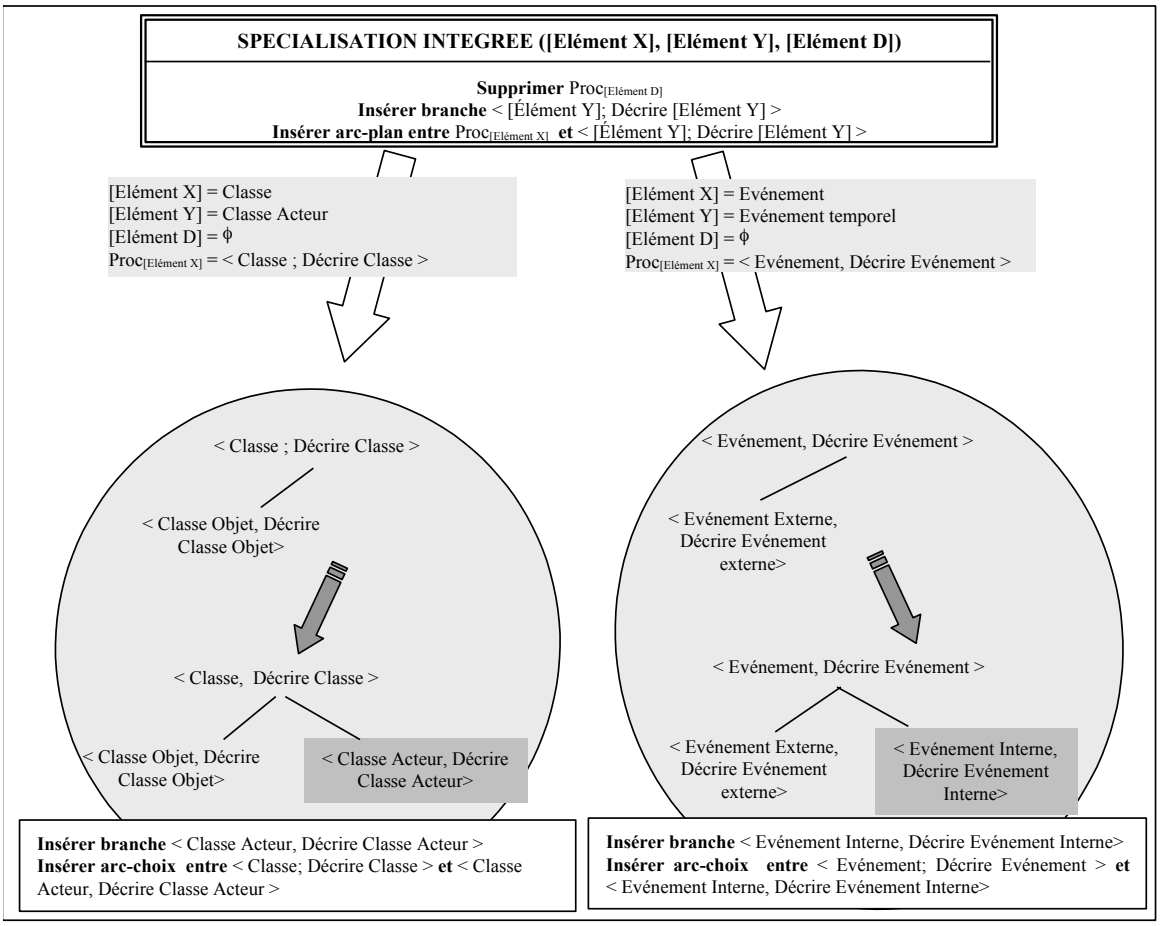


Figure 121: Exemples d'application de la stratégie SPECIALISATION INTEGREE (par type)

Le premier exemple illustre l'extension de la **DÉMARCHE** d'une méthode par l'intégration du processus de construction du concept de *Classe Acteur*, spécialisation du concept de *Classe*. Comme il s'agit d'une spécialisation par type, il suffit d'ajouter une nouvelle possibilité au contexte permettant la description de *Classe* pour lui ajouter la description de *Classe* en tant que *Classe Acteur*. De la même manière, le deuxième exemple permet d'ajouter un nouveau choix à la démarche de description du concept d'*Événement* concernant le type particulier d'*Événement* qui est *Événement Interne*.

18.3.5 SPECIALISATION INTEGREE (par état)

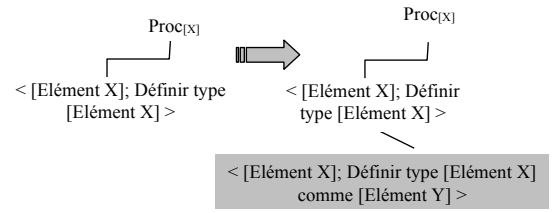
On applique cette stratégie lorsque la partie **PRODUIT** de la méthode d'origine a été étendue avec la stratégie d'extension de **PRODUIT** SPECIALISATION et que l'ingénieur de méthodes souhaite étendre la partie **DÉMARCHE** de la méthode avec la stratégie générique EXTENSION INTEGREE. Cette stratégie spécifique ne peut toutefois s'exécuter que si l'élément spécialisé possède une spécialisation par état. Si, au contraire, il a une spécialisation par type, la stratégie à appliquer sera celle de la section précédente.

18.3.5.1 Principe d'application de la stratégie SPECIALISATION INTEGREE (par état)

Si un élément de **PRODUIT** est devenu inutile du fait de l'extension de la partie **PRODUIT**, il faut supprimer sa construction dans l'arbre de processus correspondant à la **DÉMARCHE** de la méthode modifiée. Cette opération peut s'écrire de la manière suivante avec l'opérateur : ***Supprimer*** $Proc_{[Elément\ D]}$, où *[Elément D]* représente l'élément inutile et $Proc_{[Elément\ D]}$ son processus de construction.

L'application de cette stratégie spécifique permet de greffer, dans la **DÉMARCHE** de définition du type spécifique de l'élément déjà existant, une nouvelle alternative permettant de définir un nouveau type pour celui-ci (l'élément Y). Cette transformation de l'arbre de processus s'effectue donc en ajoutant une nouvelle branche et en reliant celle-ci à l'arbre. Ceci s'écrit de la façon suivante : *{* ***Insérer branche*** *< [Elément X]; Définir type [Elément X] comme [Elément Y] > ;* ***Insérer arc-plan entre*** *< [Elément X]; Définir type [Elément X] >* ***et*** *< [Elément X]; Définir type [Elément X] comme [Elément Y] >}.*

SPECIALISATION INTEGREE ([Elément X], [Elément Y], [Elément D])



Supprimer $Proc_{[Elément\ D]}$
Insérer branche < [Elément X]; Définir type [Elément X] comme [Elément Y] >
Insérer arc-choix entre < [Elément X]; Définir type [Elément X] > **et** < [Elément X]; Définir type [Elément X] comme [Elément Y] >

Figure 122 : Principe d'application de la stratégie SPECIALISATION INTEGREE (par état)

18.3.5.2 Exemples d'application de la stratégie SPECIALISATION INTEGREE (par état)

L'application de cette stratégie est illustrée dans la figure suivante sur deux exemples différents.

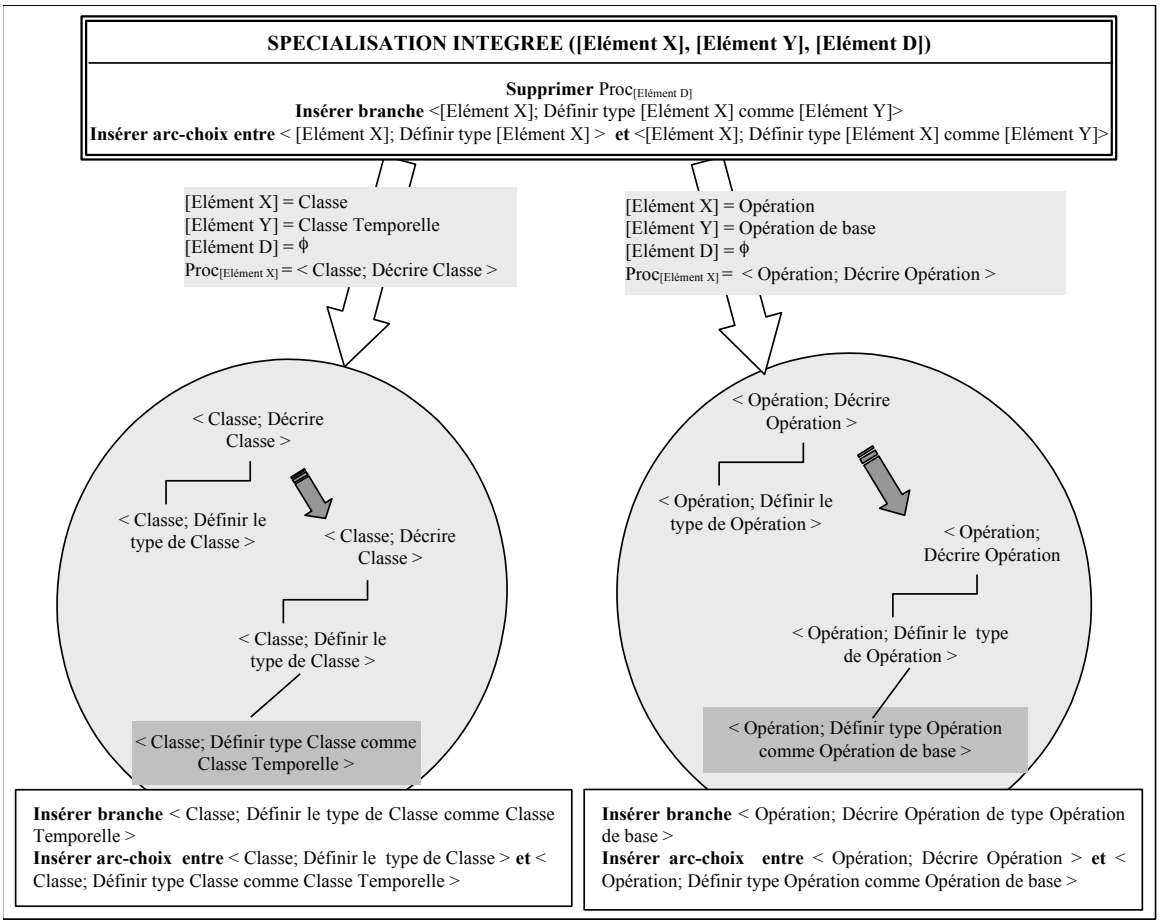


Figure 123: Exemples d'application de la stratégie SPECIALISATION INTEGREE (par état)

Ces deux exemples visualisent bien l'application de cette stratégie sur deux parties de **DÉMARCHE**s différentes. Le premier exemple permet d'intégrer un nouveau type de *Classe*, *Classe Temporelle*, dans la **DÉMARCHE** de description de ce concept. Le même principe est appliqué dans le deuxième exemple où c'est le concept d'*Opération* qui est étendu avec une nouvelle possibilité alternative pour le type *Opération de base*.

18.3.6 GENERALISATION GLOBALE (1ere Greffe)

On applique cette stratégie lorsque la partie **PRODUIT** a été étendue par la stratégie GENERALISATION, que l'ingénieur de méthodes choisit d'étendre le processus par la stratégie EXTENSION GLOBALE et dont c'est la première application sur cette méthode.

18.3.6.1 Principe d'application de la stratégie GENERALISATION GLOBALE (1ère Greffe)

Cette stratégie peut s'appliquer lorsque le **PRODUIT** a été étendu par GENERALISATION et que l'ingénieur de méthodes choisit d'étendre la **DÉMARCHE** par la stratégie EXTENSION SEQUENTIELLE GLOBALE, dans le cadre d'une première greffe de démarche avec cette stratégie.

Comme pour les stratégies précédentes, la première opération à effectuer sur la **DÉMARCHE** de la méthode est de supprimer la construction de tout élément rendu inutile par l'extension de la partie **PRODUIT**. Ceci se fait avec l'opérateur ***Supprimer*** *Proc[Elément D]*.

Qu'il n'y ait pas eu de greffe auparavant en suivant cette stratégie signifie qu'il va falloir créer une nouvelle branche de l'arbre de processus pour prendre en compte toute nouvelle extension suivant l'opérateur EXTENSION SEQUENTIELLE GLOBALE. Il faut donc ajouter une nouvelle racine à l'arbre pour mettre en séquence la construction orientée objet initiale et les modifications d'extension. La structure générique permettant d'effectuer ce traitement est représentée par la séquence d'opérateurs suivante : {***Insérer nœud*** < *Description du Problème; Construire et étendre le schéma OO* > ; ***Insérer arc-plan entre*** < *Description du Problème; Construire et étendre le schéma OO* > ***et*** *Racine (Proc[M])* }, où la fonction *Racine (Proc[M])* correspond au processus présent à la racine de l'arbre de processus et qui représente la branche permettant la construction du schéma objet avant toute extension.

Ajouter une nouvelle branche, en tant que nœud père de la racine de l'arbre de processus (donc comme nouvelle racine), insère toutes les démarches sous-jacentes. En effet, cette stratégie particulière insère le nœud racine mais également un nœud fils spécifique aux extensions (appelé <*Schéma OO; Etendre le schéma OO* >) ne contenant après l'exécution de cette stratégie qu'une seule démarche étant *< [Élément X]; Étendre, par GENERALISATION GLOBALE (1ère Greffe), [Élément X] pour intégrer [Élément Y] >*.

La dernière étape consiste à construire le graphe de précédence de cette séquence, ce qui se fait grâce à l'opérateur ***Insérer Graphe-Précédence pour*** < *Description du Problème; Construire et étendre le schéma 00* >.

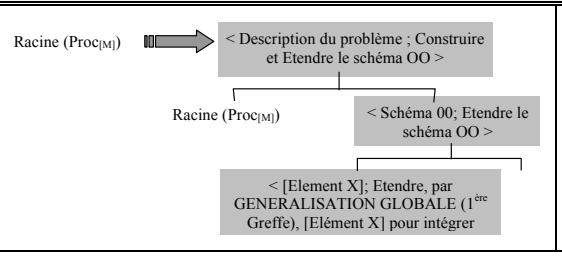
GENERALISATION GLOBALE (1ère Greffe) ([Elément X], [Elément Y], [Elément D])	
	**Supprimer** Proc[Elément D] **Insérer branche** < Description du Problème; Construire et étendre le schéma OO > **Insérer arc-plan entre** < Description du Problème; Construire et étendre le schéma OO > **et** Racine (Proc[M]) **Insérer Graphe-Précédence pour** < Description du Problème; Construire et étendre le schéma OO >

Figure 124 : Principe d'application de la stratégie GENERALISATION GLOBALE (1ère Greffe)

18.3.6.2 Exemples d'application de la stratégie GENERALISATION GLOBALE (1ère Greffe)

Deux exemples d'application de cette stratégie sont présentés dans la figure suivante.

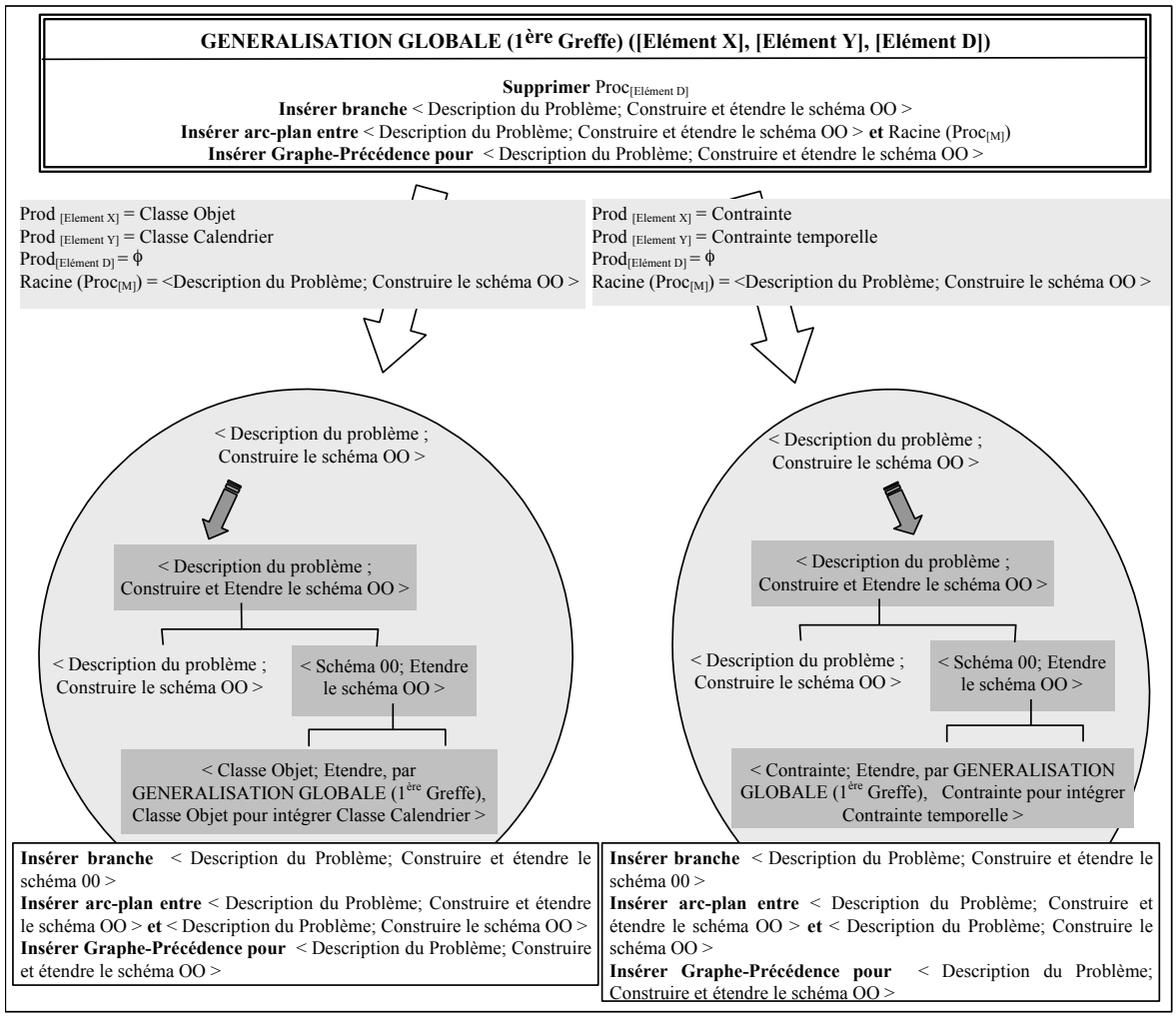


Figure 125: Exemples d'application de la stratégie GENERALISATION GLOBALE (1ère Greffe)

Les deux exemples figurant dans la figure ci-dessus illustrent l'application de cette stratégie. En effet, le premier exemple visualise la **DÉMARCHE** d'une méthode - dont on a étendu la partie **PRODUIT**, plus exactement le concept de *Classe Objet*, par GENERALISATION pour y intégrer le concept de *Classe Calendrier* – que l'on étend de façon SEQUENTIELLE GLOBALE pour y intégrer la construction de ce concept. De la même manière, le deuxième exemple présente une extension de la **DÉMARCHE** d'une méthode - dont la partie **PRODUIT** a été étendue pour intégrer le concept de *Contrainte Temporelle* - que l'on étend pour y intégrer la construction du nouveau concept.

18.3.7 GENERALISATION GLOBALE (Greffe Additionnelle)

Cette stratégie s'applique lorsque l'ingénieur de méthodes a déjà préalablement exécuté un patron d'extension de la **DÉMARCHE** avec la stratégie EXTENSION GLOBALE, que la partie **PRODUIT** a été étendue grâce à la stratégie GENERALISATION et qu'il souhaite de nouveau exécuter la stratégie d'EXTENSION SEQUENTIELLE GLOBALE.

18.3.7.1 Principe d'application de la stratégie GENERALISATION GLOBALE (Greffe Additionnelle)

Si, par l'extension de la partie **PRODUIT** de la méthode, un élément est rendu inutile, il faut également supprimer sa présence dans les **DÉMARCHE**s de l'arbre de processus. Ceci se fait par l'opérateur ***Supprimer*** $Proc_{[Elément\ D]}$.

L'ingénieur de méthodes peut se servir de cette stratégie lorsqu'il a déjà été préalablement exécuté un patron d'extension de la **DÉMARCHE** suivant la stratégie EXTENSION GLOBALE. Cela indique qu'il existe déjà une branche de l'arbre de processus qui concerne les extensions et qu'il suffit donc d'ajouter une autre démarche dans la séquence. Cette opération s'effectue par l'opérateur générique ***Insérer arc-plan entre*** *Extension (Proc[M])* ***et*** *< [Élément X] ; Étendre, par GENERALISATION GLOBALE (Greffe Additionnelle), [Élément X] pour intégrer [Élément Y]>* où *[Élément X]* représente l'élément qui était déjà présent dans la méthode d'origine, *[Élément Y]* l'élément qui a été intégré grâce à un patron d'extension du **PRODUIT** et *Extension (Proc[M])* le nœud de l'arbre représentant la **DÉMARCHE** d'extension de la méthode et regroupant en séquence toutes les constructions des concepts ajoutés lors de l'extension de la partie **PRODUIT** et que l'on souhaite intégrer dans la partie **DÉMARCHE** selon cette stratégie d'extension.

Il est ensuite nécessaire de réorganiser le graphe de transition d'états de la séquence de cette démarche car certaines extensions peuvent avoir des heuristiques de précédence entre elles. Ceci se fait avec l'opération ***Changer Graphe-Précédence pour*** *Extension (Proc[M]).*

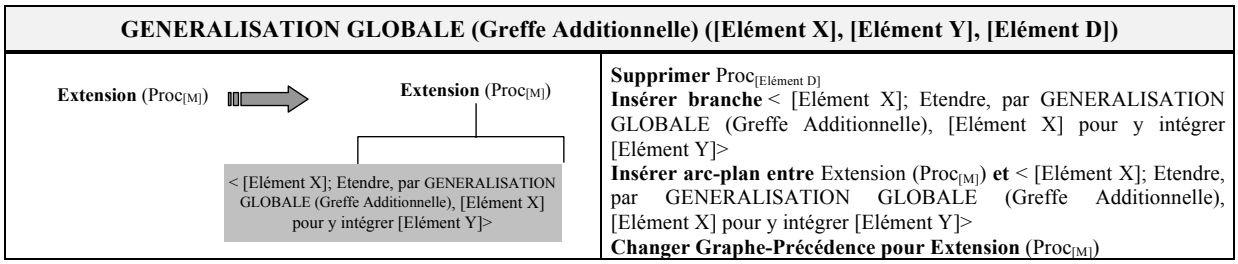


Figure 126 : Principe d'application de la stratégie GENERALISATION GLOBALE (Greffe Additionnelle)

18.3.7.2 Exemples d'application de la stratégie GENERALISATION GLOBALE (Greffe Additionnelle)

L'application de cette stratégie est illustrée par les deux exemples présentés dans la figure suivante.

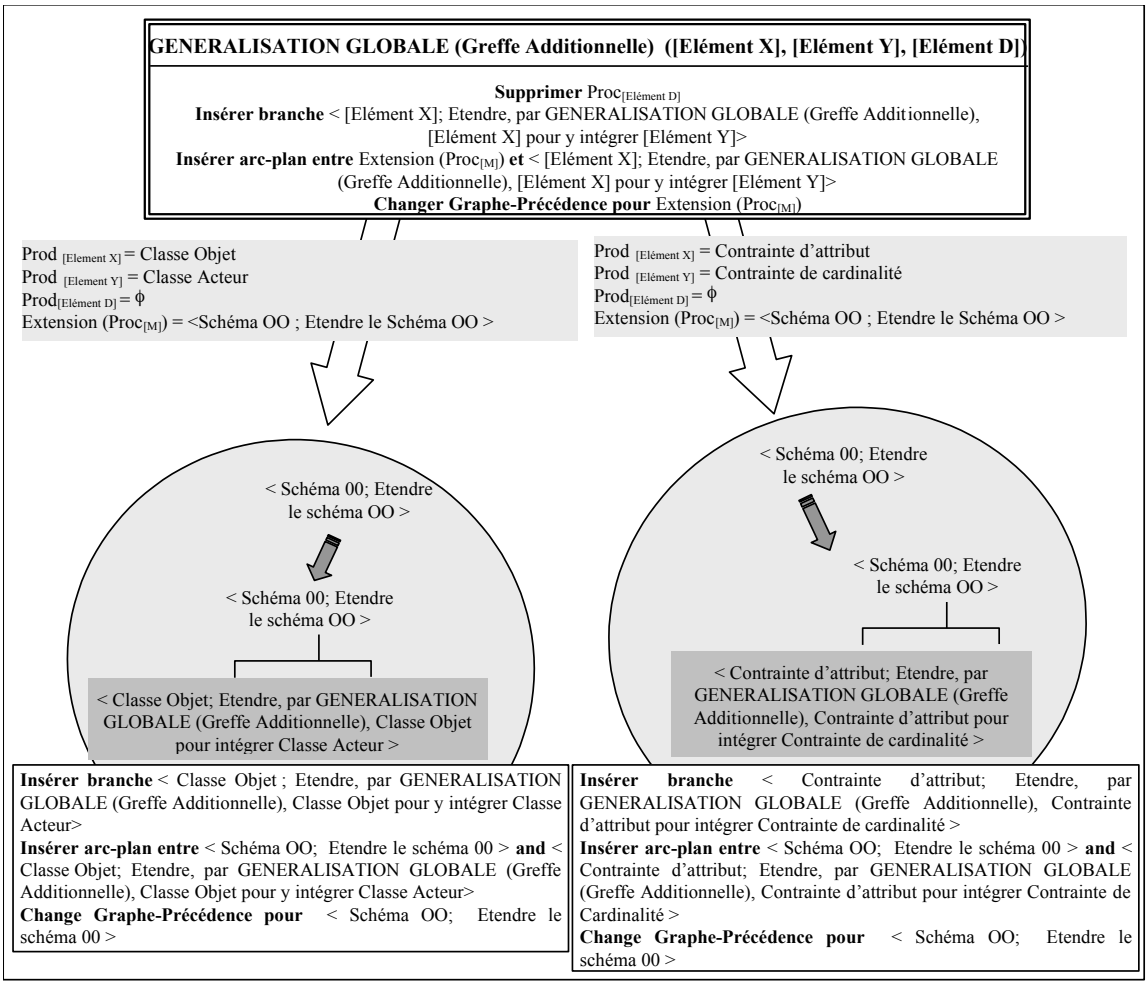


Figure 127: Exemples d'application de la stratégie GENERALISATION GLOBALE (Greffe Additionnelle)

Ces exemples illustrent deux méthodes ayant déjà été étendues auparavant par la stratégie d'extension de **DÉMARCHE** EXTENSION GLOBALE. En effet, leurs parties **DÉMARCHE** possèdent chacune un nœud représentant l'extension séquentielle globale de la méthode. Dans le premier exemple, la méthode est étendue en ajoutant la **DÉMARCHE** de construction du concept de *Classe Acteur* par généralisation de celui de *Classe Objet* en concept de *Classe*, alors que dans le second exemple, c'est la construction du concept de *Contrainte de Cardinalité* qui est intégrée par la généralisation de *Contrainte d'attribut* en *Contrainte*.

18.3.8 GENERALISATION LOCALE

On applique cette stratégie lorsque la partie **PRODUIT** a été étendue par la stratégie GENERALISATION et que l'ingénieur de méthodes choisit d'étendre la partie **DÉMARCHE** par la stratégie d'EXTENSION SEQUENTIELLE LOCALE.

18.3.8.1 Principe d'application de la stratégie GENERALISATION LOCALE

L'insertion d'un nouvel élément dans la partie **PRODUIT** de la méthode d'origine peut avoir pour conséquence le fait que l'un des éléments de la méthode devienne caduc. L'extension du **PRODUIT** prend cet état de fait en compte et donne la possibilité de supprimer cet élément. L'extension de la

partie **DÉMARCHE** doit donc proposer de supprimer la construction de cet élément dans la **DÉMARCHE** de la méthode. Ceci s'effectue avec l'opérateur générique ***Supprimer*** $Proc_{[Elément\ D]}$.

Cette stratégie greffe une démarche intermédiaire entre le processus de description de l'élément que l'on veut étendre - l'Élément X - et son nœud père. Ce processus intermédiaire permettra de mettre en séquence ce processus de description et un nouveau processus d'extension de ce même élément. La recherche d'une structure générique permettant d'effectuer ce traitement nous a conduit à l'élaboration des opérateurs suivants.

L'opérateur ***Supprimer arc entre*** $Proc_{[Elément\ X]}$ ***et*** *Parent-de* $(Proc_{[Elément\ X]})$ permet de couper le lien entre le processus de description de l'Elément X et son processus père.

La greffe du processus intermédiaire se formalise avec l'opérateur: ***Insérer nœud*** *< [Élément X]; Décrire et étendre [Élément X] >*. Ensuite, il est nécessaire de définir le type de l'arc entre la démarche de description de X et son père (plan ou choix), pour insérer le nœud intermédiaire avec un arc du bon type. Ceci se fait avec les opérateurs :

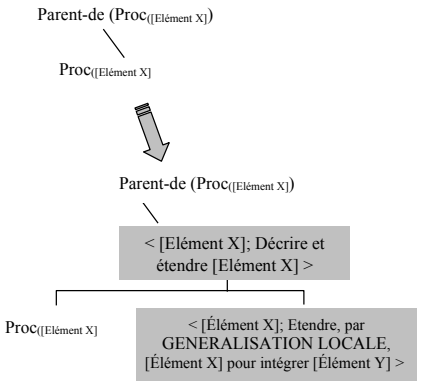
Si *Type-Parent-de* $(Proc_{[Elément\ X]})$ *= PLAN* ***alors***
Insérer arc-plan entre *< [Élément X]; Décrire et Étendre [Élément X] >* ***et***
Parent-de $(Proc_{[Elément\ X]})$
sinon
Insérer arc-choix entre *< [Élément X]; Décrire et Étendre [Élément X] >* ***et***
Parent-de $(Proc_{[Elément\ X]})$
fsi.

On insère ensuite l'arc de type plan permettant de relier ce processus intermédiaire et la démarche de description de l'élément X avec l'opérateur ***Insérer arc-plan entre*** *< [Élément X]; Décrire et Étendre [Élément X] >* ***et*** $Proc_{[Elément\ X]}$.

Les opérateurs *{* ***Insérer branche*** *< [Élément X]; Etendre, par GENERALISATION LOCALE, [Élément X] pour intégrer [Élément Y] > ;* ***Insérer arc-plan entre*** *< [Élément X]; Décrire et Étendre [Élément X] >* ***et*** *< [Élément X]; Etendre, par GENERALISATION LOCALE, [Élément X] pour intégrer [Élément Y] > }* permettent de mettre en séquence la démarche de description et la démarche d'extension.

La dernière étape consiste à construire le graphe de précédence de cette séquence d'extension, ce qui se fait grâce à l'opérateur générique ***Insérer Graphe-Précédence pour*** *< [Élément X]; Décrire et Étendre [Élément X] >*.

GENERALISATION LOCALE ([Elément X], [Elément Y], [Elément D])



Supprimer Proc[Elément D]
Supprimer arc entre Proc [Elément X] **et** Parent-de (Proc [Elément X])
Insérer nœud < [Elément X]; Décrire et étendre [Elément X] >
Si Type-Parent-de (Proc[Elément X]) = PLAN **alors**
Insérer arc-plan entre < [Élément X]; Décrire et Étendre [Élément X] > **et** Parent-de (Proc[Elément X])
sinon
Insérer arc-choix entre < [Élément X]; Décrire et Étendre [Élément X] > **et** Parent-de (Proc[Elément X])
fsi
Insérer arc-plan entre < [Elément X]; Décrire et Etendre [Elément X] > **et** Proc [Elément X]
Insérer branche < [Élément X]; Etendre, par GENERALISATION LOCALE, [Élément X] pour intégrer [Élément Y] >
Insérer arc-plan entre < [Elément X]; Décrire et Etendre [Elément X] > **et** < [Élément X]; Etendre, par GENERALISATION LOCALE, [Élément X] pour intégrer [Élément Y] >
Insérer Graphe-Précédence pour < [Elément X]; Décrire et Etendre [Elément X] >

Figure 128 : Principe d'application de la stratégie GENERALISATION LOCALE

18.3.8.2 Exemples d'application de la stratégie GENERALISATION LOCALE

La figure suivante illustre l'application de cette stratégie sur deux exemples d'extension.

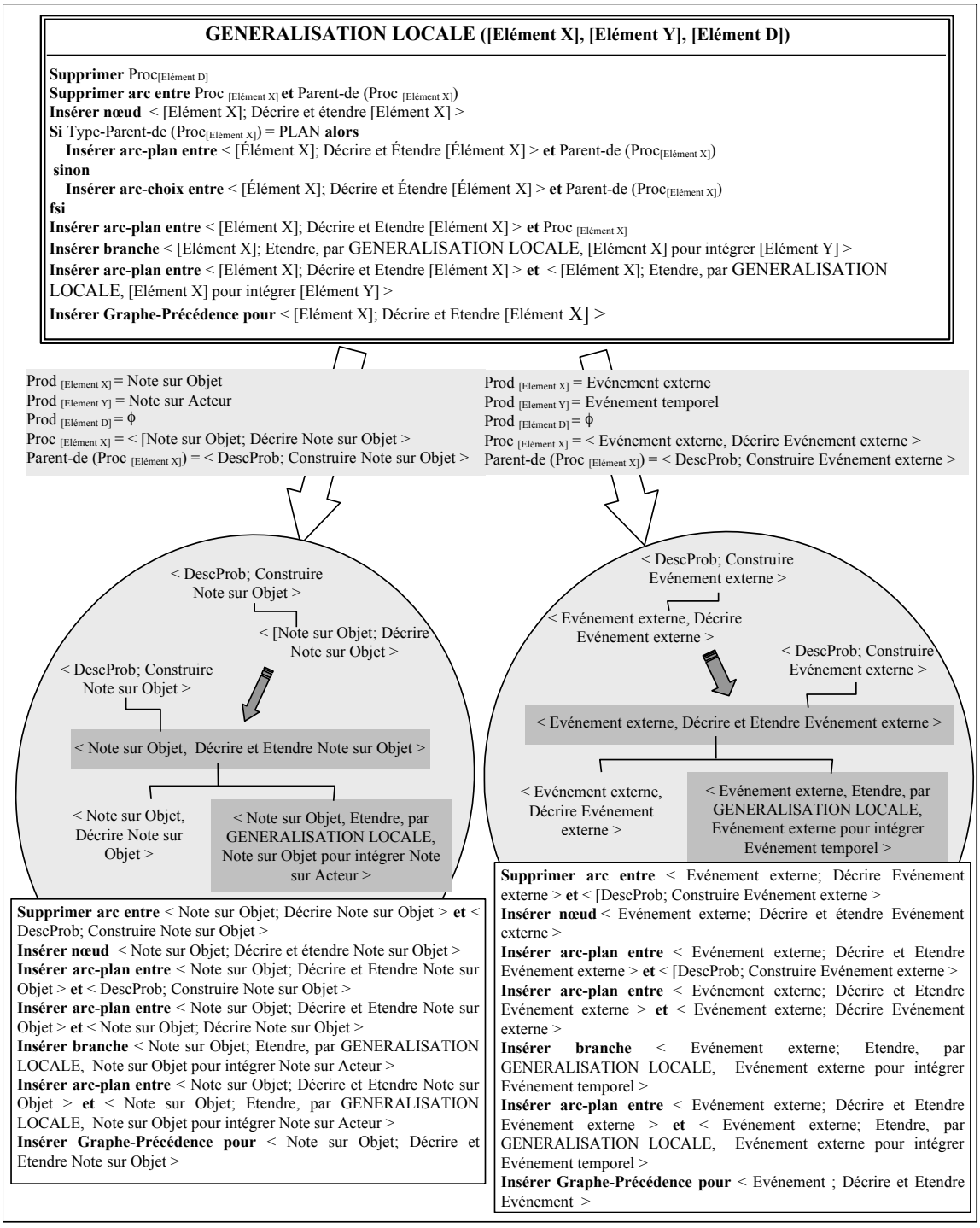


Figure 129: Exemples d'application de la stratégie GENERALISATION LOCALE

La figure précédente présente deux exemples d'application d'une EXTENSION SEQUENTIELLE LOCALE de la **DÉMARCHE** d'une méthode dont la partie **PRODUIT** a été étendue par GENERALISATION. Dans le premier exemple, c'est le concept de *Note sur Acteur* qui a été intégré dans la partie **PRODUIT** de la méthode. La partie **DÉMARCHE** est ici étendue de manière locale dans le but d'y intégrer la construction de ce concept. De la même façon, le deuxième exemple intègre la construction du concept d'*Evénement Temporel*.

18.3.9 GENERALISATION INTEGREE (par type)

On applique cette stratégie lorsque la partie **PRODUIT** de la méthode d'origine a été étendue avec la stratégie d'extension de **PRODUIT** GENERALISATION et que l'ingénieur de méthodes souhaite étendre la partie **DÉMARCHE** de la méthode avec la stratégie générique d'EXTENSION INTEGREE. Cette stratégie spécifique ne peut toutefois s'exécuter que si l'élément généralisé ne fait une distinction entre ses spécialisations que par type. Si, au contraire, il a une spécialisation par état, la stratégie à appliquer sera celle de la section suivante.

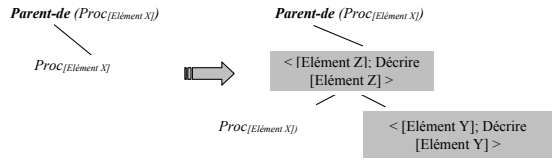
18.3.9.1 Principe d'application de la stratégie GENERALISATION INTEGREE (par type)

Comme pour les autres stratégies d'extension de la **DÉMARCHE**, il est parfois nécessaire de supprimer la construction d'un élément devenu inutile lors de l'extension de la partie **PRODUIT**. Cette opération peut s'écrire grâce à l'opérateur ***Supprimer*** $Proc_{[Elément D]}$.

Pour intégrer la construction d'un élément que l'on a intégré par généralisation, il suffit d'ajouter un autre père au processus de description de l'élément que l'on a généralisé pour avoir deux alternatives : l'une permettant la description déjà existante préalablement et l'autre permettant celle du nouvel élément. La structure générique de ce traitement est représentée par la séquence d'opérateurs suivante : { ***Supprimer arc*** $Proc_{[Élément X]}$ ***et Parent-de*** $Proc_{[Élément X]}$; ***Renommer*** $Proc_{[Élément X]}$ ***avec*** *[Nom* $Proc_{[Elément X]}$*]* ; ***Insérer branche*** < *[Elément Z]; Décrire [Elément Z]* > où la fonction *Parent-de* $Proc_{[Élément X]}$ correspond au nœud père de la démarche de description de l'Élément X ($Proc_{[Élément X]}$) et *[Nom* $Proc_{[Elément X]}$*]* représente le nouveau nom qui est associé à cette démarche (dans le cas où le fait de renommer est souhaitable pour la compréhension de la **DÉMARCHE**.
Il est ensuite nécessaire de définir si l'arc entre la démarche de description de X et son père est un plan ou un choix, de manière à insérer la démarche intermédiaire avec un arc du bon type. Ceci se fait avec la séquence d'opérateurs suivante :

Si *Type-Parent-de (*$Proc_{[Elément X]}$*) = PLAN* ***alors***
Insérer arc-plan entre *Parent-de* $Proc_{[Élément X]}$ ***et*** *< [Elément Z]; Décrire [Elément Z] >*
sinon
Insérer arc-choix entre *Parent-de* $Proc_{[Élément X]}$ ***et*** *< [Elément Z]; Décrire [Elément Z] >*
fsi.
Puis on rattache la nouvelle démarche à la démarche de description de l'élément X avec l'opérateur ***Insérer arc-choix entre*** *< [Elément Z]; Décrire [Elément Z] >* ***et*** $Proc_{[Élément X]}$ }.

GENERALISATION INTEGREE ([Elément X], [Elément Y], [Elément D], [Nom X])



Supprimer Proc[Elément D]
Supprimer arc entre Proc[Elément X] **et Parent-de** (Proc[Elément X])
Renommer Proc[Élément X] **avec** [Nom Proc[Elément X]]
Insérer branche < [Élément Z]; Décrire [Élément Z]>
Si Type-Parent-de (Proc[Elément X]) = PLAN **alors**
Insérer arc-plan entre Parent-de Proc[Élément X] **et** < [Elément Z]; Décrire [Elément Z] >
sinon
Insérer arc-choix entre Parent-de Proc[Élément X] **et** < [Elément Z]; Décrire [Elément Z] >
fsi
Insérer arc-choix entre <[Élément Z]; Décrire [Élément Z]> **et** Proc[Elément X]

Figure 130 : Principe d'application de la stratégie GENERALISATION INTEGREE (par type)

18.3.9.2 Exemples d'application de la stratégie GENERALISATION INTEGREE (par type)

La figure suivante présente deux exemples qui illustrent l'application de cette stratégie.

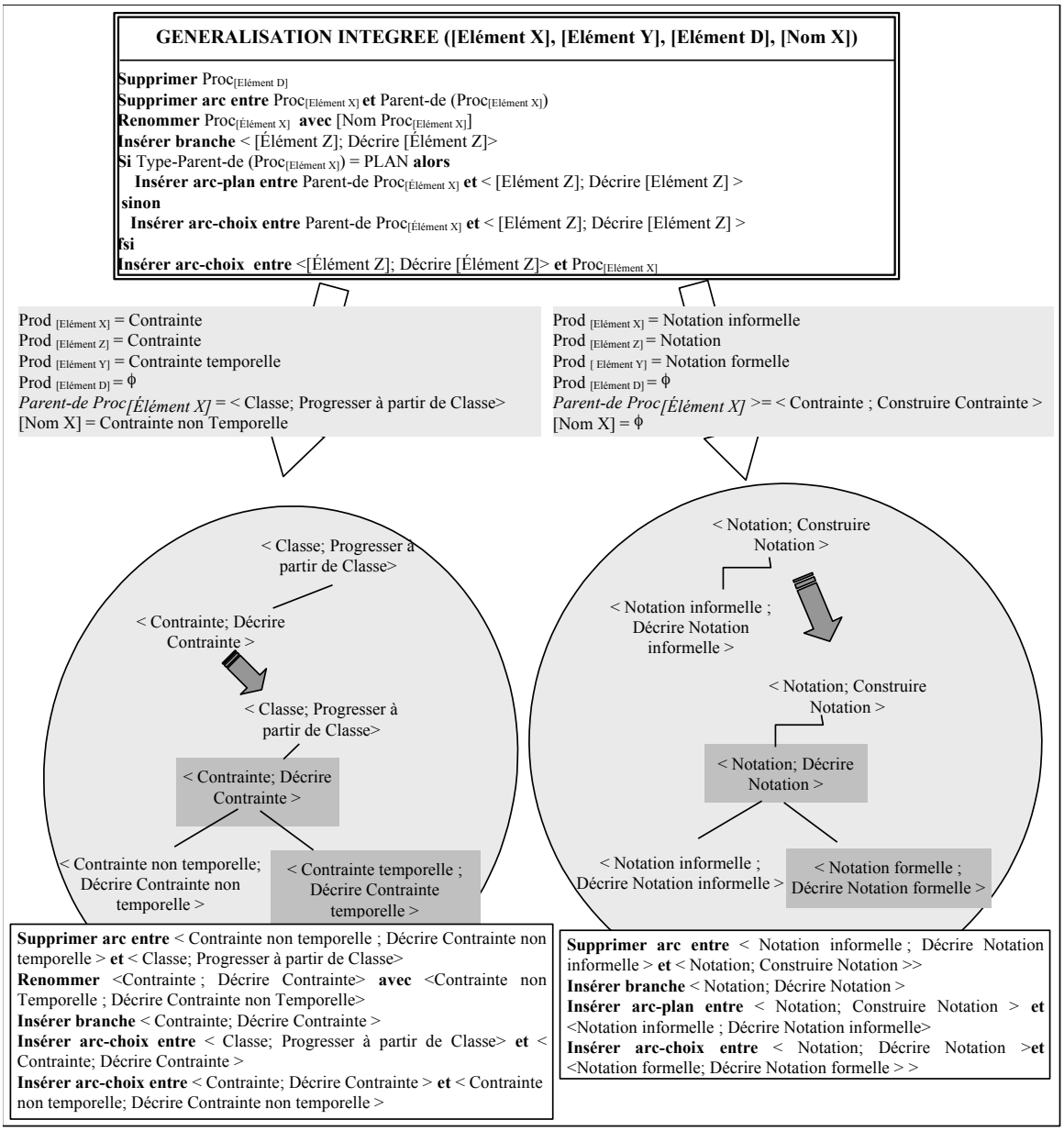


Figure 131: Exemples d'application de la stratégie GENERALISATION INTEGREE (par type)

Le premier exemple de la figure précédente illustre l'extension de la **DÉMARCHE** d'une méthode que l'on étend de manière INTEGREE pour y ajouter la construction du concept de *Contrainte Temporelle* alors que le **PRODUIT** de cette méthode a été étendu par GENERALISATION. On peut visualiser ici le nouveau nom qui est associé à la démarche de description des *Contraintes* : la description des *Contraintes non Temporelles*. Ensuite l'insertion d'un nœud supplémentaire entre la démarche de *progression à partir de Classe* et la démarche de *Description d'une Contrainte non Temporelle* qui permet de généraliser cette dernière pour pouvoir intégrer la démarche de *Description d'une Contrainte Temporelle*. De la même manière le deuxième exemple présente l'insertion d'un nœud intermédiaire entre la démarche de *Construction de Notation* et celle de *Description de Notation Informelle* pour permettre l'insertion d'une autre alternative qui sera celle de la *Description de Notation Formelle*.

18.3.10 GENERALISATION INTEGREE (par état)

Cette stratégie s'applique si la partie **PRODUIT** de la méthode d'origine a été étendue avec la stratégie d'extension de **PRODUIT** par GENERALISATION et que l'ingénieur de méthodes souhaite étendre la partie **DÉMARCHE** avec la stratégie d'EXTENSION INTEGREE. Toutefois, il ne peut exécuter cette stratégie que si l'élément généralisé ne fait une distinction entre ses spécialisations que par état. Si, au contraire, il a une spécialisation par type, la stratégie à appliquer sera celle de la section précédente.

18.3.10.1 Principe d'application de la stratégie GENERALISATION INTEGREE (par état)

Ainsi que pour les stratégies précédentes, et pour les mêmes raisons, l'application d'une extension peut rendre caduc un élément présent préalablement dans la partie **PRODUIT** de la méthode. Comme les extensions de **PRODUIT** prennent en compte cette possibilité et autorise l'ingénieur de méthodes à supprimer cet élément, il est nécessaire, pour la cohérence de la méthode concernée, de supprimer également sa démarche de description dans la partie **DÉMARCHE**. L'opérateur permettant de supprimer cette démarche spécifique est défini de la manière suivante : ***Supprimer*** $Proc_{[Elément\ D]}$ où *[Elément D]* représente l'élément rendu inutile par l'extension et $Proc_{[Elément\ D]}$ sa démarche de description.

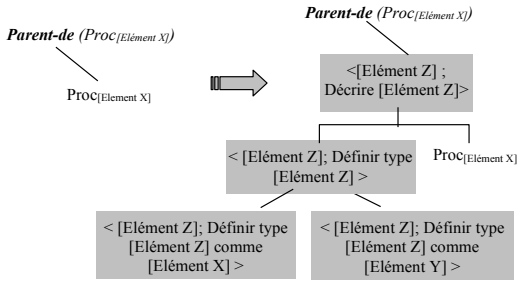
On insère ensuite une branche intermédiaire entre la démarche de description de l'[Elément X] et son nœud père, ce qui se fait avec les opérateurs suivants : *{**Supprimer arc entre*** $Proc_{[Elément\ X]}$ ***et*** *Parent-de (*$Proc_{[Elément\ X]}$*) ;* ***Renommer*** $Proc_{[Élément\ X]}$ ***avec*** *[Nom* $Proc_{[Elément\ X]}$*] ;* ***Insérer branche*** *< [Élément Z]; Décrire [Élément Z]>*. Il est à noter que l'opérateur permettant de renommer la démarche de description de l'élément X n'est pas obligatoire mais n'est utilisé que lorsque ce nom peut être raffiné pour permettre une meilleure compréhension de la démarche.
Il est ensuite nécessaire de définir si l'arc entre la démarche de description de X et son père est un plan ou un choix, de manière à insérer la démarche intermédiaire avec un arc du bon type. Ceci se fait avec la séquence d'opérateurs suivante :

Si *Type-Parent-de (*$Proc_{[Elément\ X]}$*) = PLAN* ***alors***
 Insérer arc-plan entre *Parent-de* $Proc_{[Élément\ X]}$ ***et*** *< [Elément Z]; Décrire [Elément Z] >*
sinon
 Insérer arc-choix entre *Parent-de* $Proc_{[Élément\ X]}$ ***et*** *< [Elément Z]; Décrire [Elément Z] >*
fsi.

Puis on rattache la nouvelle démarche avec la démarche de description de l'élément X avec l'opérateur ***Insérer arc-plan entre*** *<[Élément Z]; Décrire [Élément Z]>* ***et*** $Proc_{[Elément\ X]}$*>}*. Ces opérateurs permettent de mettre en séquence la démarche de définition du type de l'élément et sa démarche de description.

Pour finir, il est nécessaire de définir l'ordre de précédence dans lequel doivent s'exécuter ces deux démarches, ce qui se fait avec *l'opérateur* ***Insérer Graphe-Précédence pour*** *<[Élément Z]; Décrire [Élément Z]>*.

GENERALISATION INTEGREE ([Elément X], [Elément Y], [Elément D], [Nom X])



Supprimer Proc[Elément D]
Supprimer arc entre Proc[Elément X] **et** Parent-de (Proc[Elément X])
Renommer Proc[Élément X] **avec** [Nom Proc[Elément X]]
Insérer branche < [Élément Z]; Décrire [Élément Z]>
Si Type-Parent-de (Proc[Elément X]) = PLAN **alors**
Insérer arc-plan entre Parent-de Proc[Élément X] **et** < [Elément Z]; Décrire [Elément Z] >
sinon
Insérer arc-choix entre Parent-de Proc[Élément X] **et** < [Elément Z]; Décrire [Elément Z] >
fsi
Insérer arc-plan entre <[Élément Z]; Décrire [Élément Z]> **et** Proc[Elément X]
Insérer Graphe-Précédence pour <[Élément Z]; Décrire [Élément Z]>

Figure 132 : Principe d'application de la stratégie GENERALISATION INTEGREE (par état)

18.3.10.2 Exemples d'application de la stratégie GENERALISATION INTEGREE (par état)

La figure suivante illustre l'application de cette stratégie sur deux exemples d'extension.

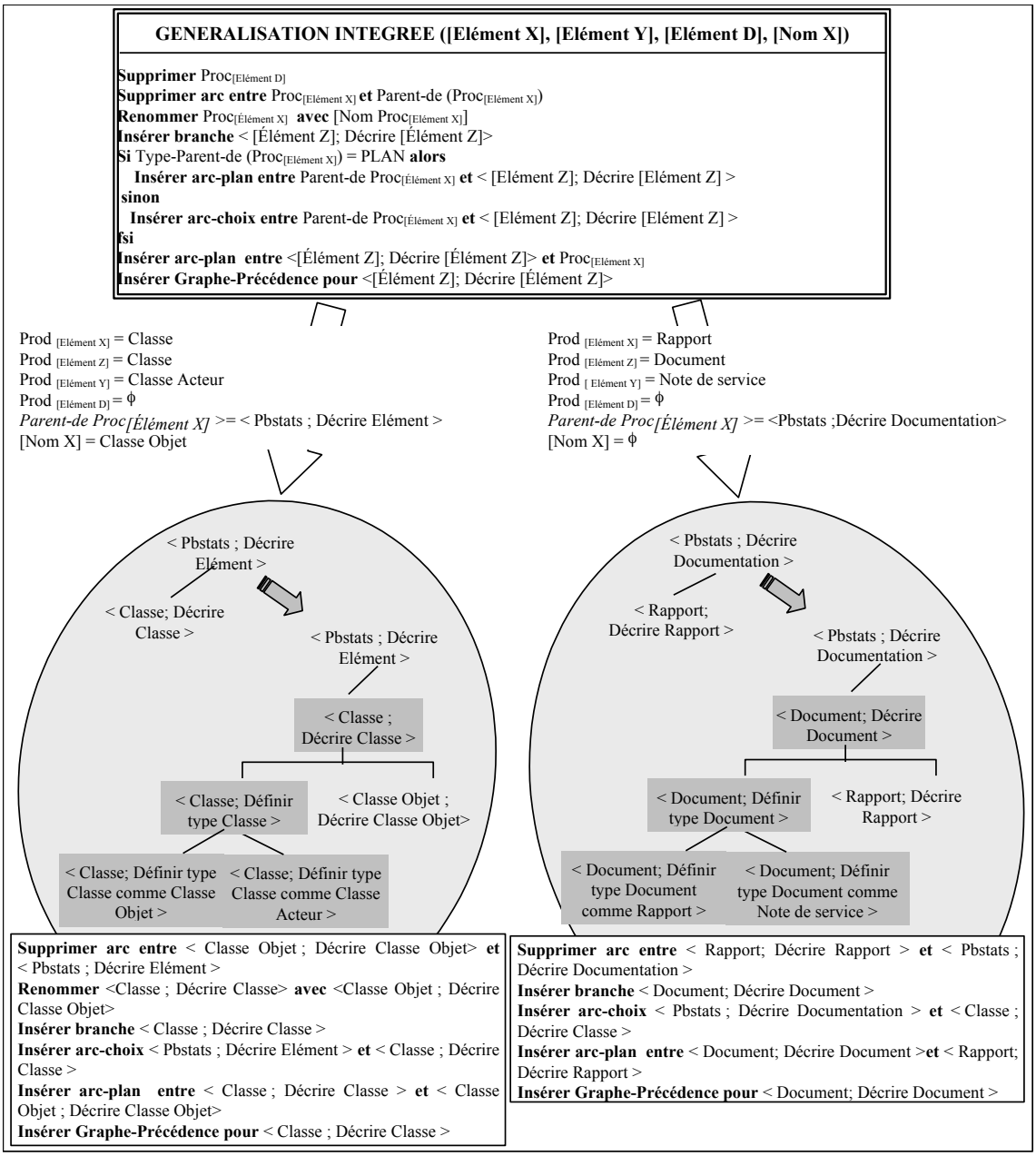


Figure 133: Exemples d'application de la stratégie GENERALISATION INTEGREE (par état)

Ces deux exemples visualisent les opérateurs à appliquer lors de l'exécution de cette stratégie. Une généralisation d'un concept a été faite lors de l'extension de la partie **PRODUIT** dans le but d'offrir une nouvelle spécialisation (l'élément que l'on a inséré par l'extension du **PRODUIT**) à cette généralisation. Le choix de l'EXTENSION INTEGREE comme stratégie d'extension de la **DÉMARCHE** de la méthode permet d'insérer, de façon transparente dans l'arbre de processus, les démarches permettant de gérer ces généralisations. Dans le premier exemple, le concept de *Classe* est renommer avec celui de *Classe Objet* qui est ensuite généralisé pour pouvoir insérer celui de *Classe Acteur* alors que dans le second exemple c'est le concept de *Rapport* qui est généralisé pour insérer celui de *Note de Service*. L'extension de la **DÉMARCHE** se fait donc simplement en greffant une branche intermédiaire entre la démarche de description de l'élément que l'on généralise et son père dans l'arbre.

18.3.11 COMPOSITION GLOBALE (1ere Greffe)

On applique cette stratégie lorsque la partie **PRODUIT** a été étendue par la stratégie COMPOSITION et que l'ingénieur de méthodes choisit d'étendre la partie **DÉMARCHE** par la stratégie d'EXTENSION SEQUENTIELLE GLOBALE (1ère Greffe).

18.3.11.1 Principe d'application de la stratégie COMPOSITION GLOBALE (1ère Greffe)

La première opération à effectuer correspondant à cet opérateur est, bien entendu, celle qui permet de supprimer la démarche de construction de l'élément devenu inutile du fait de l'extension de la partie **PRODUIT**, si tel est le cas. Cette modification s'effectue avec l'opérateur suivant : ***Supprimer*** *Proc[Elément D]*.

Le fait qu'il n'y ait pas eu de greffe préalable suivant cette stratégie particulière signifie qu'il va falloir créer une nouvelle branche de l'arbre de processus pour prendre en compte toute nouvelle extension. Il faut ajouter une nouvelle racine à l'arbre pour mettre en séquence la construction orientée objet initiale et les modifications d'extension. La recherche d'une structure générique permettant d'effectuer ce traitement nous a conduit à l'élaboration de la séquence d'opérateurs suivante : {***Insérer nœud*** < *Description du Problème; Construire et étendre le schéma OO* > ; ***Insérer arc-plan entre*** < *Description du Problème; Construire et étendre le schéma OO* > ***et*** *Racine (Proc[M])* }, où la fonction *Racine (Proc[M])* correspond à la démarche présente à la racine de l'arbre de processus et qui représente la branche permettant la construction du schéma objet avant toute extension.

Insérer cette nouvelle racine à l'arbre de processus insère aussi toutes ses démarches sous-jacentes. En effet, cette stratégie particulière insère également un nœud fils spécifique aux extensions (appelé <*Schéma OO; Etendre le schéma OO* >) ne contenant après l'exécution de cette stratégie qu'une seule démarche étant < *[Élément X]; Etendre, par COMPOSITION GLOBALE (1ère Greffe), [Élément X] pour intégrer [Élément Y]* >.

La dernière étape consiste à construire le graphe de précédence de cette séquence, ce qui se fait grâce à l'opérateur ***Insérer Graphe-Précédence pour*** < *Description du Problème; Construire et étendre le schéma 00* >.

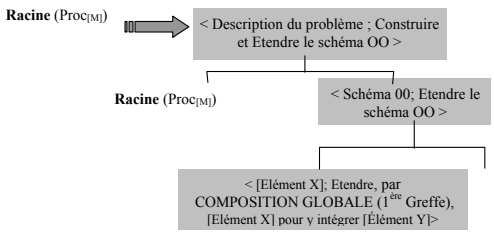
COMPOSITION GLOBALE (1ère Greffe) ([Elément X], [Elément Y], [Elément D])	
	**Supprimer** Proc[Elément D] **Insérer branche** < Description du Problème; Construire et étendre le schéma OO > **Insérer arc-plan entre** < Description du Problème; Construire et étendre le schéma OO > **et** Racine (Proc[M]) **Insérer Graphe-Précédence pour** < Description du Problème; Construire et étendre le schéma OO >

Figure 134 : Principe d'application de la stratégie COMPOSITION GLOBALE (1ère Greffe)

18.3.11.2 Exemples d'application de la stratégie COMPOSITION GLOBALE (1ère Greffe)

La figure suivante illustre l'application de cette stratégie à l'aide de deux exemples.

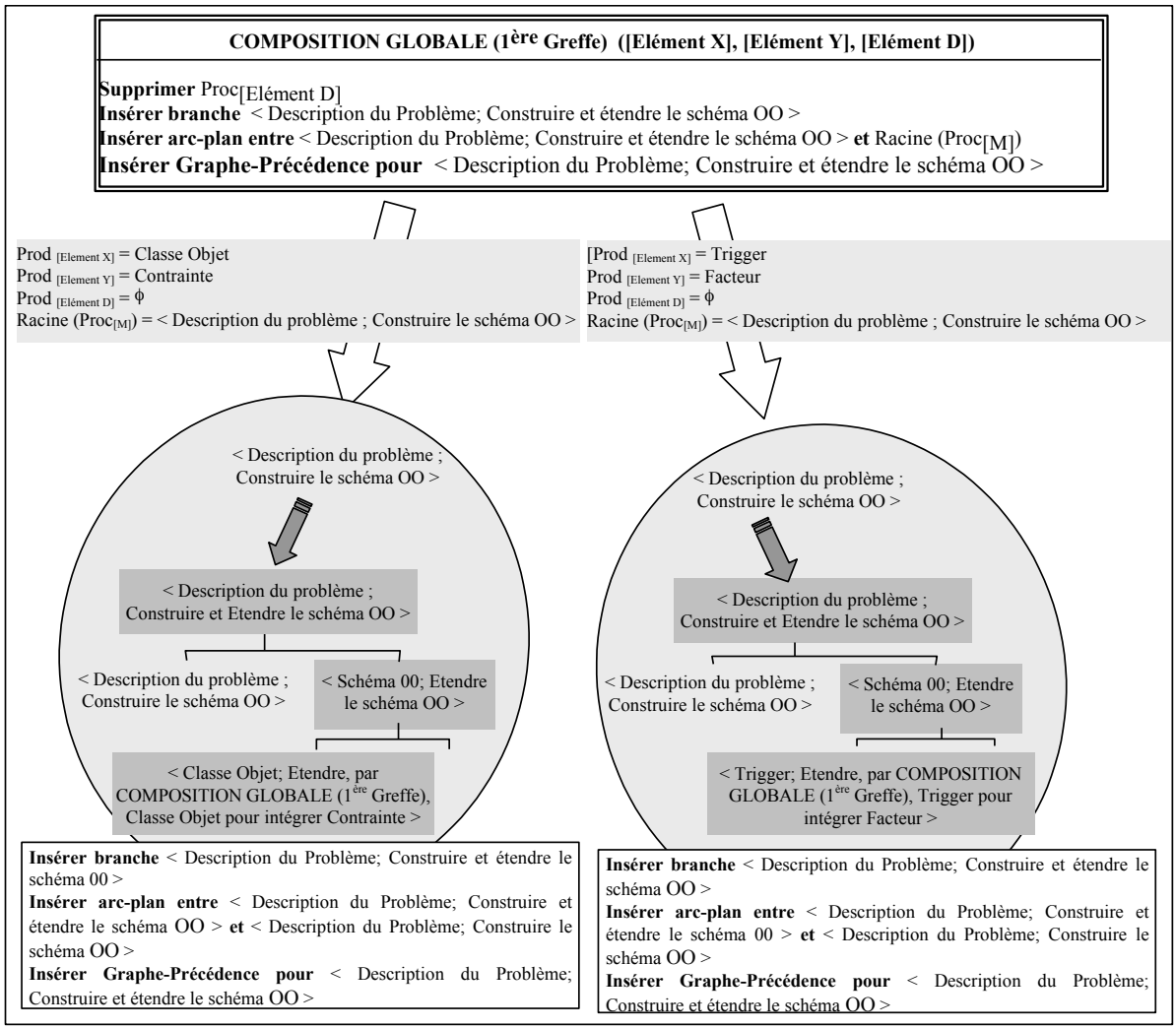


Figure 135: Exemples d'application de la stratégie COMPOSITION GLOBALE (1ère Greffe)

Les exemples visualisés dans la figure précédente montrent l'application de cette stratégie sur des méthodes dont le **PRODUIT** a été étendu par COMPOSITION et dont l'ingénieur de méthodes souhaite étendre la **DÉMARCHE** de façon SEQUENTIELLE GLOBALE. Comme c'est la première application de cette stratégie d'extension de **DÉMARCHE** pour ces méthodes, l'extension intègre dans les deux cas une nouvelle racine à l'arbre de processus dans le but de mettre en séquence la construction originelle du schéma OO et les extensions possibles qui lui seront rattachées. Dans le premier cas, c'est l'insertion de la construction du concept de *Contrainte* qui est inséré par l'extension comme composé de celui de *Classe Objet*, alors que dans le second cas, c'est la construction du concept de *Facteur* qui est rattaché à celui de *Trigger*.

18.3.12 COMPOSITION GLOBALE (Greffe Additionnelle)

Cette stratégie s'applique lorsque l'ingénieur de méthodes a déjà préalablement exécuté un patron d'extension de la **DÉMARCHE** avec la stratégie EXTENSION SEQUENTIELLE GLOBALE, que la partie **PRODUIT** a été étendue grâce à la stratégie COMPOSITION et qu'il souhaite de nouveau exécuter la stratégie d'EXTENSION SEQUENTIELLE GLOBALE.

18.3.12.1 Principe d'application de la stratégie COMPOSITION GLOBALE (Greffe Additionnelle)

La première étape de cette extension est de supprimer toute trace de la construction des éléments qui ont pu être supprimés lors de l'extension de la partie **PRODUIT**. Ceci se fait avec l'opérateur ***Supprimer*** $Proc_{[Elément\ D]}$ où *[Elément D]* représente cet élément supprimé.

Comme il a déjà préalablement été exécuté un patron d'extension de la **DÉMARCHE** suivant la stratégie d'EXTENSION SEQUENTIELLE GLOBALE, il existe déjà une branche de l'arbre de processus qui concerne les extensions suivant cette stratégie. Il suffit donc d'ajouter une nouvelle **DÉMARCHE** dans la séquence pour prendre en compte l'extension qui nous intéresse. Cet ajout se fait grâce à l'opérateur générique ***Insérer arc-plan entre*** *Extension (Proc[M])* ***et*** *< [Élément X] ; Étendre, par COMPOSITION GLOBALE (Greffe Additionnelle), [Élément X] pour intégrer [Élément Y]>* où *[Élément X]* représente l'élément qui était déjà présent dans la méthode d'origine, *[Élément Y]* l'élément qui a été intégré grâce à un patron d'extension du **PRODUIT** et *Extension (Proc[M])* la branche de l'arbre représentant les extensions.

Il est ensuite nécessaire de réorganiser le graphe de transition d'états de la séquence de ce processus car certaines extensions peuvent avoir des heuristiques de précédence entre elles. Ceci se fait avec l'opération ***Changer Graphe-Précédence pour*** *Extension (Proc[M]).*

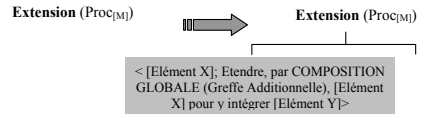
COMPOSITION GLOBALE (Greffe Additionnelle) ([Elément X], [Elément Y], [Elément D])	
	**Supprimer** $Proc_{[Elément\ D]}$ **Insérer branche** < [Elément X]; Etendre, par COMPOSITION GLOBALE (Greffe Additionnelle), [Elément X] pour y intégrer [Elément Y]> **Insérer arc-plan entre** Extension ($Proc_{[m]}$) **et** < [Elément X]; Etendre, par COMPOSITION GLOBALE (Greffe Additionnelle), [Elément X] pour y intégrer [Elément Y]> **Change Graphe-Précédence pour** Extension ($Proc_{[m]}$)

Figure 136 : Principe d'application de l'opérateur COMPOSITION GLOBALE(Greffe Additionnelle)

18.3.12.2 Exemples d'application de la stratégie COMPOSITION GLOBALE (Greffe Additionnelle)

La figure ci-dessous illustre l'application de cette stratégie par la présentation de deux exemples.

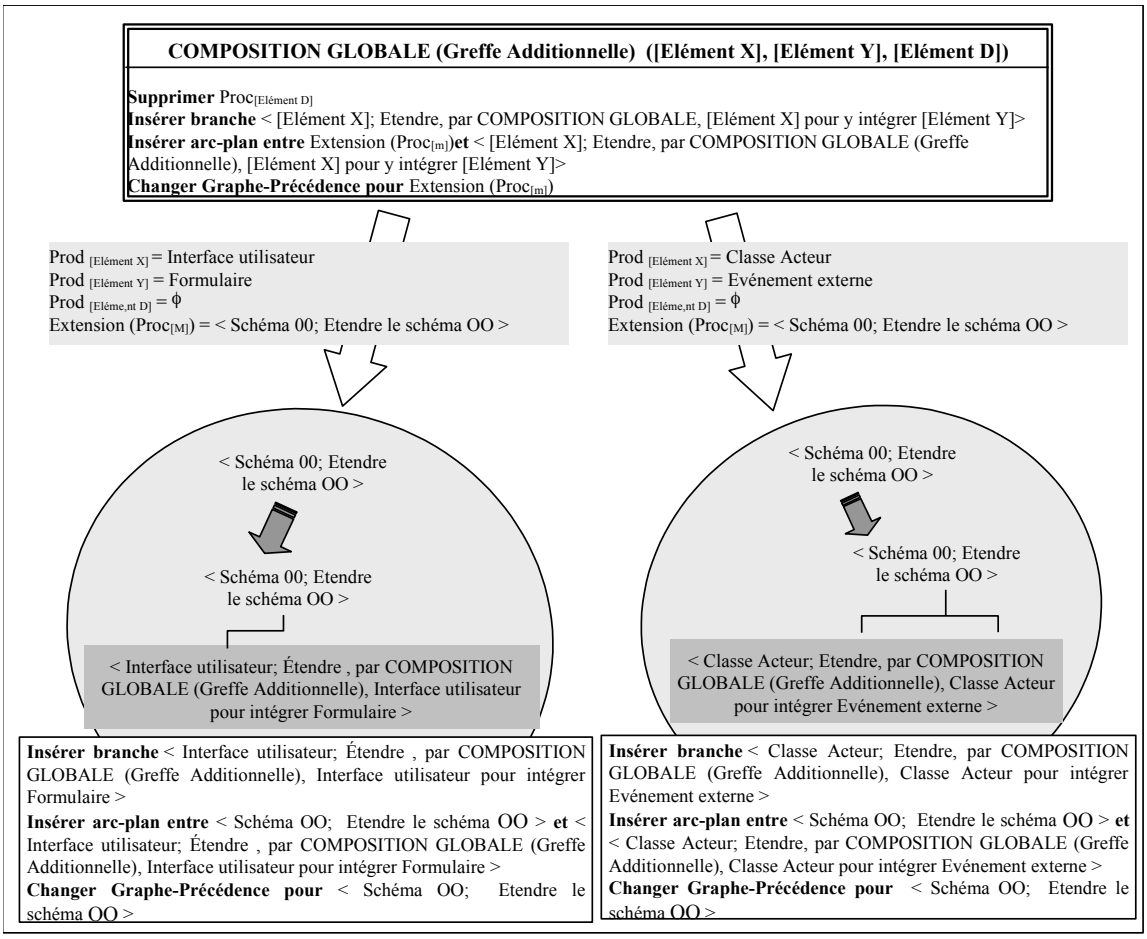


Figure 137: Exemples d'application de la stratégie COMPOSITION GLOBALE (Greffe Additionnelle)

Les deux exemples précédents illustrent deux extensions de méthodes ayant été préalablement étendues par COMPOSITION (pour la partie **PRODUIT**) et dont l'ingénieur de méthodes souhaite étendre la partie **DÉMARCHE** avec la stratégie d'EXTENSION SEQUENTIELLE GLOBALE. Comme ces deux méthodes ont déjà été étendues avec cette stratégie d'extension de **DÉMARCHE**, il n'est pas nécessaire, comme pour la stratégie précédente, d'insérer une nouvelle racine permettant la mise en séquence de la construction du schéma OO et la démarche d'extension de celui-ci. En effet, ces deux contextes sont déjà présents dans l'arbre de processus. Il suffit donc de greffer une nouvelle branche dans la séquence de cette démarche pour intégrer la construction du nouveau concept inséré dans le **PRODUIT**. Dans le premier cas, il s'agit du concept de *Formulaire* qui est un composé de l'*Interface Utilisateur*, et dans le deuxième cas, il s'agit du concept d'*Evénement Externe* qui est, lui, un composé de *Acteur*.

18.3.13 COMPOSITION LOCALE

On applique cette stratégie lorsque la partie **PRODUIT** a été étendue par la stratégie COMPOSITION et que l'ingénieur de méthodes choisit d'étendre la partie **DÉMARCHE** par la stratégie d'EXTENSION SEQUENTIELLE LOCALE.

18.3.13.1 Principe d'application de la stratégie COMPOSITION LOCALE

Comme pour les stratégies précédentes, la première opération de transformation d'une stratégie d'extension du **PRODUIT** étant une suppression d'un possible élément de **PRODUIT** rendu caduc par l'extension de la méthode, il faut, lors de l'extension de la partie **DÉMARCHE**, supprimer toute trace de cet élément. Il est donc nécessaire de supprimer la **DÉMARCHE** de construction de cet élément dans l'arbre de processus. Cet opérateur s'écrit de la manière suivante : ***Supprimer*** $Proc_{[Elément\ D]}$, où *[Elément D]* représente l'élément inutile et $Proc_{[Elément\ D]}$ son processus de construction.

L'application de cette stratégie permet de greffer un processus intermédiaire entre le processus de description de l'élément que l'on veut étendre - $Proc_{[Elément\ X]}$ - et son nœud père dans l'arbre de processus - *Parent-de* ($Proc_{[Elément\ X]}$). Ce processus intermédiaire permettra de mettre en séquence la **DÉMARCHE** de description et la nouvelle **DÉMARCHE** d'extension de ce même élément. La recherche d'une structure générique permettant d'effectuer ce traitement nous a conduit à l'élaboration des opérateurs suivants.

L'opérateur ***Supprimer arc entre*** $Proc_{[Elément\ X]}$ ***et Parent-de*** ($Proc_{[Elément\ X]}$) permet de couper le lien entre la démarche de description de l'Elément X et son noeud père.

L'opérateur permettant de greffer le processus intermédiaire se formalise de la façon suivante : ***Insérer nœud*** *< [Élément X]; Décrire et étendre [Élément X] >* . Ensuite, pour définir si l'arc entre la démarche de description de X et son père est un plan ou un choix et de manière à insérer le processus intermédiaire avec un arc du bon type, nous avons la séquence d'opérateurs suivante :

Si *Type-Parent-de* ($Proc_{[Elément\ X]}$) *= PLAN* ***alors***
Insérer arc-plan entre *< [Élément X]; Décrire et Étendre [Élément X] >* ***et***
Parent-de ($Proc_{[Elément\ X]}$)
sinon
Insérer arc-choix entre *< [Élément X]; Décrire et Étendre [Élément X] >* ***et***
Parent-de ($Proc_{[Elément\ X]}$)
fsi.

On insère ensuite l'arc de type plan permettant de relier ce processus intermédiaire et la démarche de description de l'élément X avec l'opérateur ***Insérer arc-plan entre*** *< [Élément X]; Décrire et Étendre [Élément X] >* ***et*** $Proc_{[Elément\ X]}$.

Les opérateurs *{* ***Insérer branche*** *< [Élément X]; Etendre, par COMPOSITION LOCALE, [Élément X] pour intégrer [Élément Y] > ;* ***Insérer arc-plan entre*** *< [Élément X]; Décrire et Étendre [Élément X] >* ***et*** *< [Élément X];]; Etendre, par COMPOSITION LOCALE, [Élément X] pour intégrer [Élément Y] >}* permettent de mettre en séquence les démarches de construction et d'extension.

Il est ensuite nécessaire d'organiser le graphe de transition d'états de cette séquence grâce à l'opérateur ***Insérer Graphe-Précédence pour*** *< [Élément X]; Décrire et Étendre [Élément X] >*.

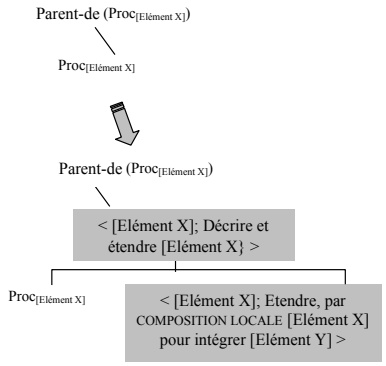
COMPOSITION LOCALE ([Elément X], [Elément Y], [Elément D])	
	**Supprimer** Proc[Elément D] **Supprimer arc entre** Proc[Elément X] **et** Parent-de (Proc[Elément X]) **Insérer nœud** < [Elément X]; Décrire et étendre [Elément X] > **Si** Type-Parent-de (Proc[Elément X]) = PLAN **alors** **Insérer arc-plan entre** < [Élément X]; Décrire et Étendre [Élément X] > **et** Parent-de (Proc[Elément X]) **sinon** **Insérer arc-choix entre** < [Élément X]; Décrire et Étendre [Élément X] > **et** Parent-de (Proc[Elément X]) **fsi** **Insérer arc-plan entre** < [Elément X]; Décrire et Etendre [Elément X] > **et** Proc[Elément X] **Insérer branche** < [Elément X]; Etendre, par COMPOSITION LOCALE [Elément X] pour intégrer [Elément Y] > **Insérer arc-plan entre** < [Elément X]; Décrire et Etendre [Elément X] > **et** < [Elément X]; Etendre, par COMPOSITION LOCALE [Elément X] pour intégrer [Elément Y] > **Insérer Graphe-Précédence pour** < [Elément X]; Décrire et Etendre [Elément X] >

Figure 138 : Principe d'application de la stratégie COMPOSITION LOCALE

18.3.13.2 Exemples d'application de la stratégie COMPOSITION LOCALE

La figure suivante illustre l'application de cette stratégie spécifique sur deux exemples d'extension.

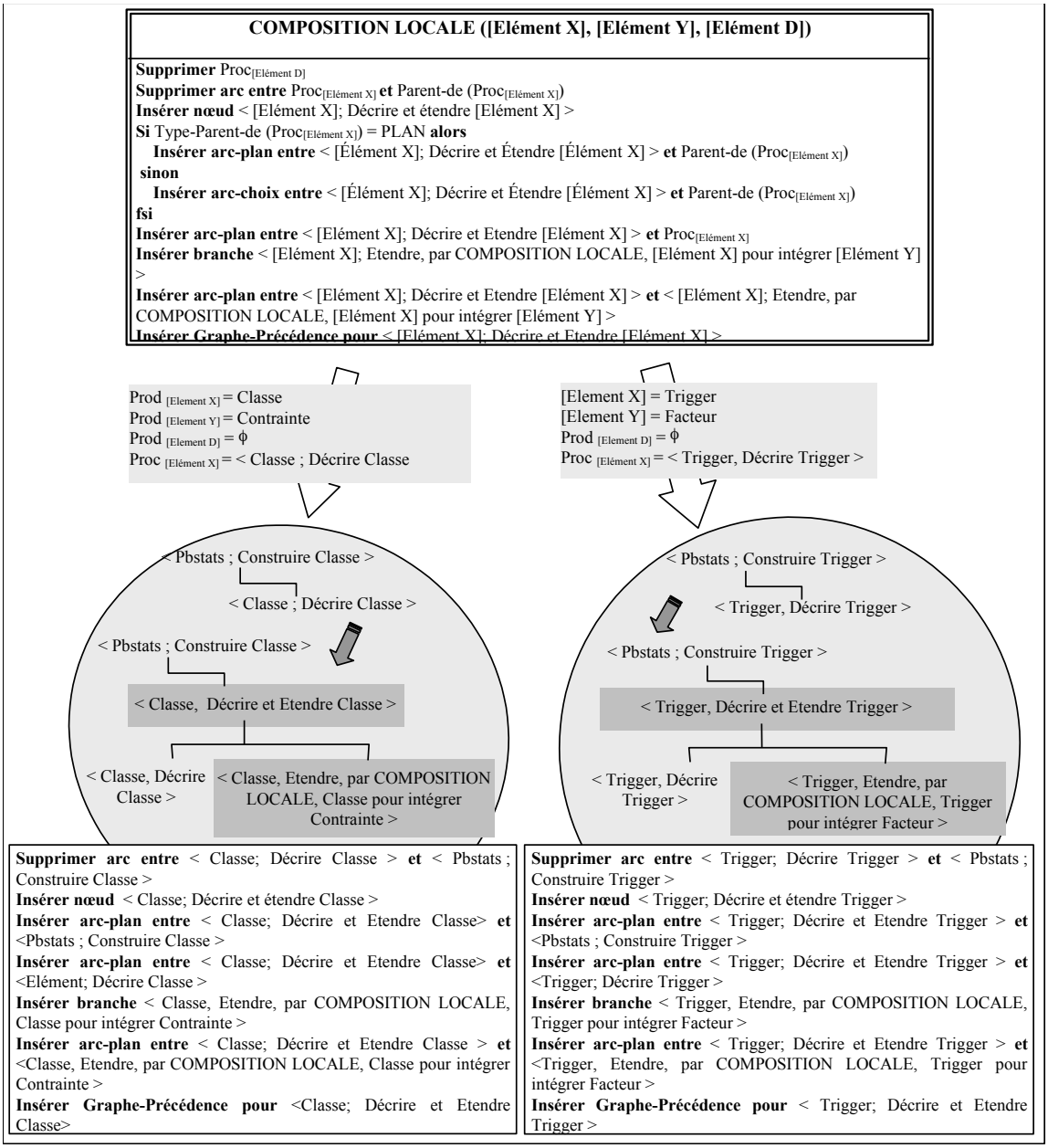


Figure 139: Exemples d'application de la stratégie COMPOSITION LOCALE

Les deux exemples précédents présentent l'application de cette stratégie sur deux méthodes ayant donc été étendues par COMPOSITION en ce qui concerne la partie **PRODUIT** et que l'ingénieur de méthodes souhaite étendre en suivant la stratégie d'EXTENSION SEQUENTIELLE LOCALE en ce qui concerne la partie **DÉMARCHE**. On peut voir ici qu'il suffit d'insérer un nœud supplémentaire entre le nœud de description de l'élément composant et son père, ce qui permettra de mettre en séquence cette démarche de description et la démarche d'extension permettant de lui adjoindre un composé. Dans le premier cas, il s'agit du concept de *Contrainte* que l'on ajoute à celui de *Classe* alors que dans le second cas il s'agit du concept de *Facteur* que l'on ajoute à celui de *Trigger*.

18.3.14 COMPOSITION INTEGREE

On applique cette stratégie lorsque la partie **PRODUIT** de la méthode d'origine a été étendue avec la stratégie d'extension de **PRODUIT** COMPOSITION et que l'ingénieur de méthodes souhaite étendre la partie **DÉMARCHE** de la méthode avec la stratégie générique d'EXTENSION INTEGREE.

18.3.14.1 Principe d'application de la stratégie COMPOSITION INTEGREE

Pour la même raison que pour les autres stratégies, il est utile de supprimer toute trace d'un élément rendu inutile dans l'arbre de processus (suppression de la **DÉMARCHE** de construction de cet élément). Cet opérateur peut s'écrire de la manière suivante : ***Supprimer*** $Proc_{[Elément\ D]}$, où *[Elément D]* représente l'élément inutile et $Proc_{[Elément\ D]}$ son processus de construction.

L'application de cette stratégie spécifique permet de greffer, à la séquence de la **DÉMARCHE** de description d'un élément déjà existant, un nouvelle branche correspondant à la **DÉMARCHE** de construction de l'élément intégré lors de l'extension de la partie **PRODUIT**. Cette transformation de l'arbre de processus s'effectue donc en ajoutant une nouvelle branche et en reliant celle-ci à l'arbre. Ceci s'écrit de la façon suivante : *{**Insérer branche** < [Élément Y]; Décrire [Elément Y] > ; **Insérer arc-plan entre** $Proc_{[Elément\ X]}$ **et** < [Élément Y]; Décrire [Elément Y] }*.

De plus, il est nécessaire de réorganiser la séquence de la démarche de description de l'[Elément X] puisqu'il est possible qu'il y ait des relations de précédence entre les anciens nœuds et celui que l'on vient de greffer. Ceci se fait avec l'opérateur ***Changer Graphe-Précédence pour*** $Proc_{[Element\ X]}$.

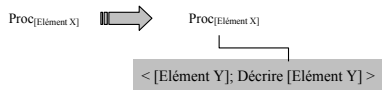
COMPOSITION INTEGREE ([Elément X], [Elément Y], [Elément D])	
	**Supprimer** Proc[Elément D] **Insérer branche** <[Elément Y] ; Décrire [Elément Y]> **Insérer Arc-plan entre** Proc[Elément X] **et** <[Elément Y] ; Décrire [Elément Y]> **Changer Graphe-Précédence pour** Proc[Elément X]

Figure 140 : Principe d'application de la stratégie COMPOSITION INTEGREE

18.3.14.2 Exemples d'application de la stratégie COMPOSITION INTEGREE

Deux exemples d'application de cette stratégie sont présentés à la figure suivante.

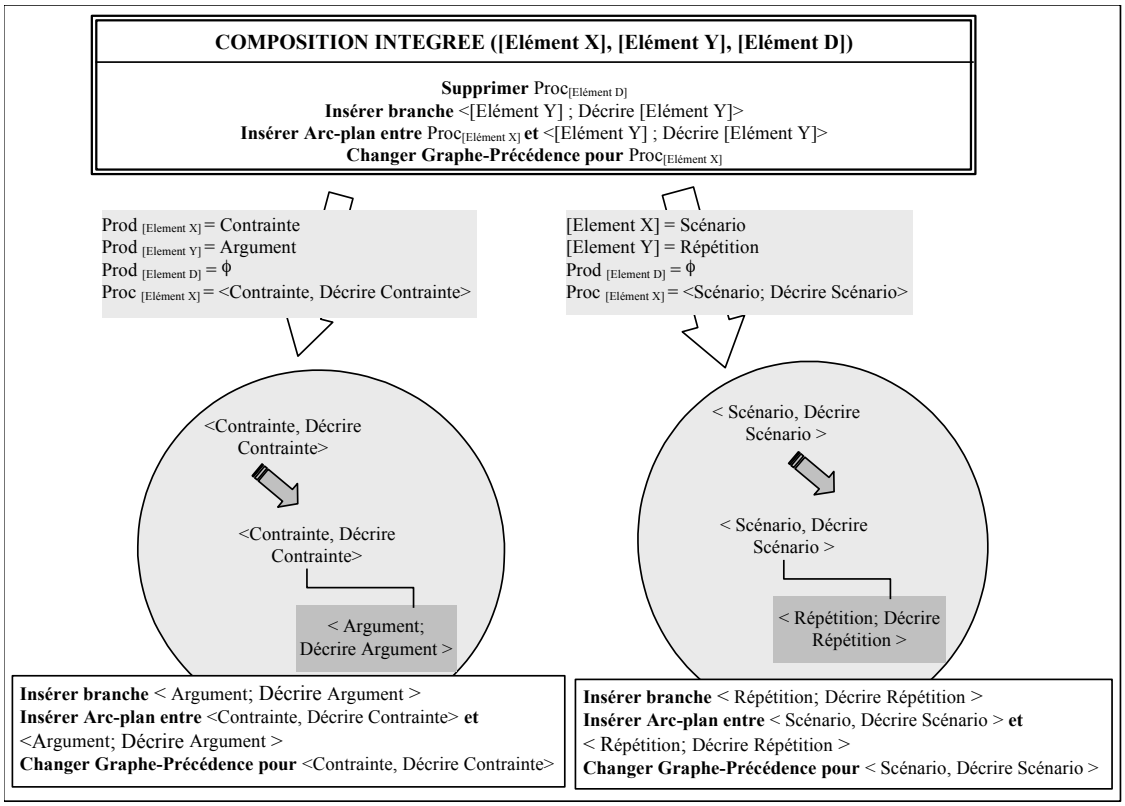


Figure 141: Exemples d'application de la stratégie COMPOSITION INTEGREE

Les deux exemples donnés à la figure précédente illustrent des applications de la stratégie d'EXTENSION INTEGREE lorsque la partie **PRODUIT** de la méthode a été étendue par COMPOSITION. Dans ce cas, il suffit de greffer une nouvelle démarche dans la séquence de description de l'élément que l'on compose qui concerne la démarche de description du nouvel élément composant. Dans le premier cas, c'est le concept d'*Argument* qui est ajouté à la composition de *Contrainte* alors que le second cas illustre l'ajout du concept de *Répétition* dans celui de *Scénario*.

18.3.15 DECOMPOSITION GLOBALE (1ere Greffe)

L'ingénieur de méthodes a choisi d'étendre son **PRODUIT** par la stratégie DECOMPOSITION et souhaite maintenant étendre la partie **DÉMARCHE** de sa méthode par la stratégie EXTENSION SEQUENTIELLE GLOBALE alors qu'il n'a pas encore effectué d'extension en suivant cette stratégie spécifique.

18.3.15.1 Principe d'application de la stratégie DECOMPOSITION GLOBALE (1ère GREFFE)

Comme pour les stratégies précédentes, la première opération à effectuer sur la **DÉMARCHE** de la méthode est de supprimer la construction de tout élément rendu inutile par l'extension de la partie **PRODUIT**. Ceci se fait avec l'opérateur ***Supprimer*** *Proc[Elément D]*.

Le fait qu'il n'y ait pas eu de greffe préalable suivant cette stratégie particulière signifie qu'il va falloir créer une nouvelle branche de l'arbre de processus pour prendre en compte toute nouvelle extension

suivant la stratégie d'EXTENSION SEQUENTIELLE GLOBALE. En fait, il suffit d'ajouter une nouvelle racine à l'arbre qui permettra de mettre en séquence la construction orientée objet initiale puis les modifications d'extension. La structure générique permettant d'effectuer ce traitement est la suivante : {***Insérer nœud*** < *Description du Problème; Construire et étendre le schéma OO* > ; ***Insérer arc-plan entre*** < *Description du Problème; Construire et étendre le schéma OO* > ***et*** *Racine (Proc[M])* }, où la fonction *Racine (Proc[M])* correspond au processus présent à la racine de l'arbre de processus et qui représente la branche permettant la construction du schéma objet avant toute extension.

Greffer ce nœud comme une nouvelle racine à l'arbre de processus insère aussi toutes ses démarches sous-jacentes. Cette stratégie insère donc également un nœud fils spécifique relatif aux extensions (appelé <*Schéma OO; Etendre le schéma OO* >) ne contenant après l'exécution de cette stratégie qu'une seule démarche étant < *[Élément X]; Etendre, par DECOMPOSITION GLOBALE (1ère Greffe), [Élément X] pour intégrer [Élément Y]* >.

La dernière étape consiste à construire le graphe de précédence de cette séquence, ce qui se fait grâce à l'opérateur ***Insérer Graphe-Précédence pour*** < *Description du Problème; Construire et étendre le schéma 00* >.

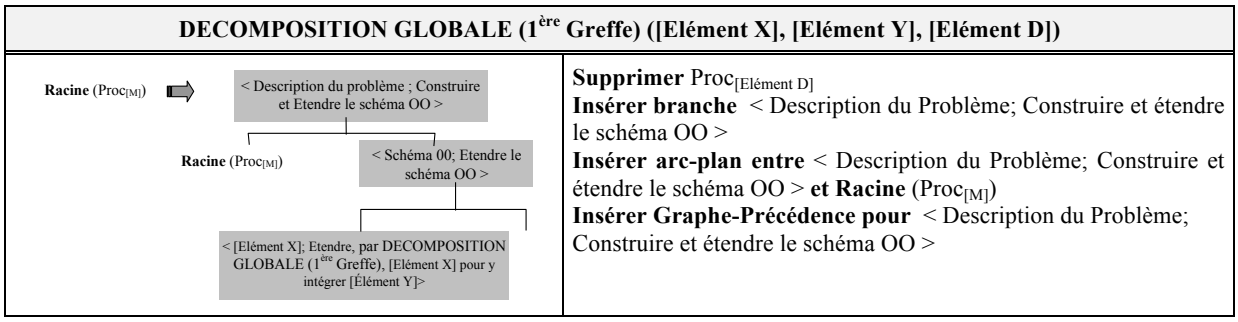


Figure 142 : Principe d'application de la stratégie DECOMPOSITION GLOBALE (1ère Greffe)

18.3.15.2 Exemples d'application de la stratégie DECOMPOSITION GLOBALE (1ère Greffe)

La figure suivante illustre l'application de cette stratégie sur deux exemples d'extension.

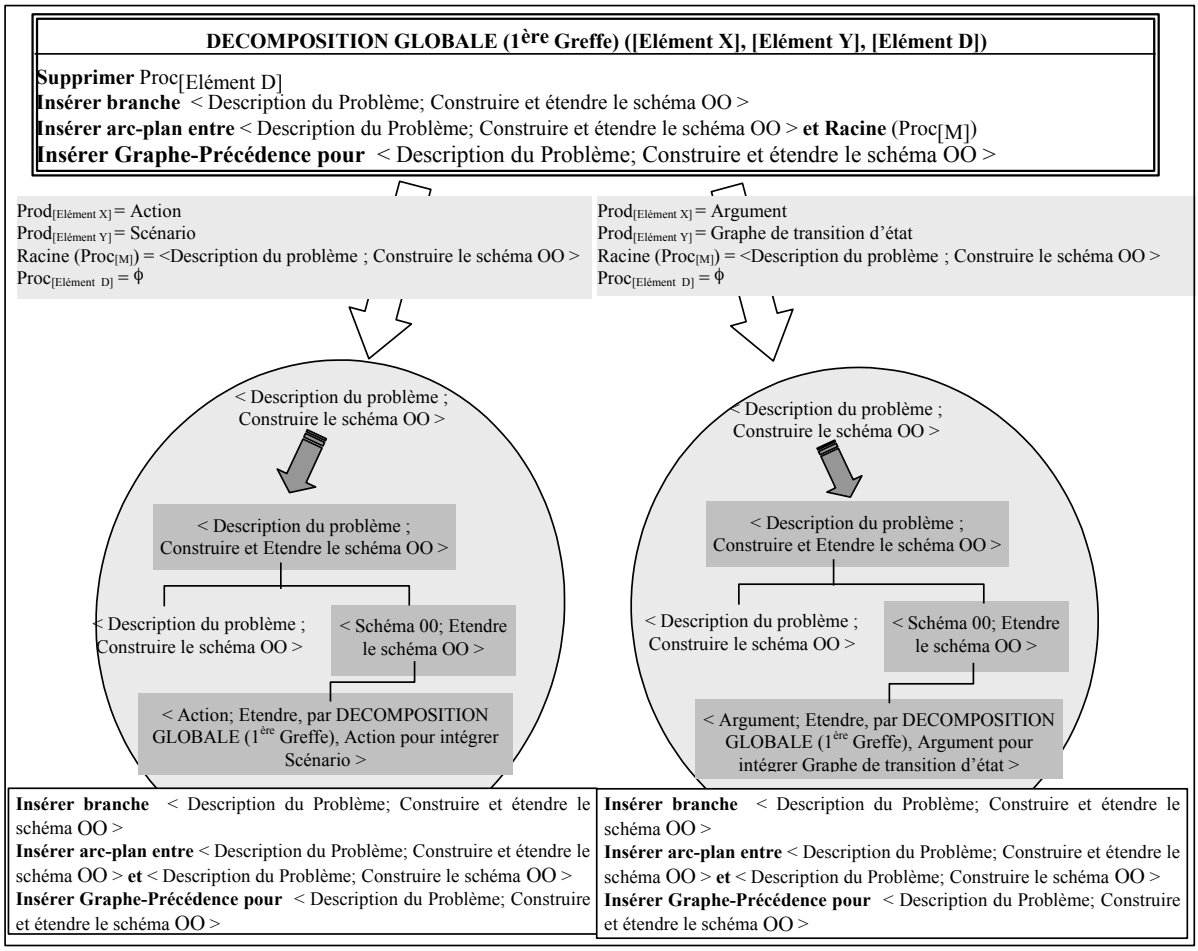


Figure 143: Exemples d'application de la stratégie DECOMPOSITION GLOBALE (1ère Greffe)

Les deux exemples de la figure précédente présentent deux applications possibles de cette stratégie d'extension de la **DÉMARCHE** sur deux méthodes ayant donc été étendue préalablement avec la stratégie d'extension DECOMPOSITION. Ces applications permettent en fait d'attribuer une nouvelle racine à l'arbre de processus dans le but de séquencer la construction du schéma OO et son extension. Dans le premier cas, on intègre par ce moyen la démarche de construction du concept de *Scénario* (composé du concept d'*Action* déjà présent dans la méthode avant extension) et dans le second cas c'est le concept de *Graphe de transition d'état* qui est intégré comme concept composé de celui d'*Argument*, concept également présent dans la méthode préalablement à toute extension.

18.3.16 DECOMPOSITION GLOBALE (Greffe Additionnelle)

Cette stratégie s'applique lorsque l'ingénieur de méthodes a déjà préalablement exécuté un patron d'extension de la **DÉMARCHE** avec la stratégie d'EXTENSION SEQUENTIELLE GLOBALE, que la partie **PRODUIT** a été étendue grâce à la stratégie DECOMPOSITION et qu'il souhaite de nouveau exécuter la stratégie d'EXTENSION SEQUENTIELLE GLOBALE.

18.3.16.1 Principe d'application de la stratégie DECOMPOSITION GLOBALE (Greffe Additionnelle)

La première modification apportée par une stratégie d'extension du **PRODUIT** est une suppression d'un possible élément de **PRODUIT** rendu caduc par l'extension de la méthode, il faut donc, lors de l'extension de la partie **DÉMARCHE**, supprimer toute trace de cet élément. Il est donc nécessaire de supprimer la démarche de construction de cet élément dans l'arbre de processus. Cet opérateur s'écrit: ***Supprimer*** $Proc_{[Elément\ D]}$, où *[Elément D]* représente l'élément inutile et $Proc_{[Elément\ D]}$ son processus de construction.

La partie **DÉMARCHE** ayant déjà été étendue avec cette stratégie, il existe déjà une branche de l'arbre de processus qui concerne les extensions suivant cette stratégie et qu'il suffit donc d'ajouter un nouveau processus dans la séquence pour prendre en compte celle qui nous intéresse. Cette opération se fait avec l'opérateur ***Insérer arc-plan entre*** *Extension (Proc[M])* ***et*** *< [Élément X] ; Etendre, par DECOMPOSITION GLOBALE (Greffe Additionnelle), [Élément X] pour intégrer [Élément Y]>* où *[Élément X]* représente l'élément qui était déjà présent dans la méthode d'origine et *[Élément Y]* l'élément qui a été intégré grâce à un patron d'extension du **PRODUIT**.

Il est ensuite nécessaire de réorganiser le graphe de transition d'états de la séquence de ce processus car certaines extensions peuvent avoir des heuristiques de précédence entre elles. Ceci se fait avec l'opération ***Changer Graphe-Précédence pour*** *Extension (Proc[M]).*

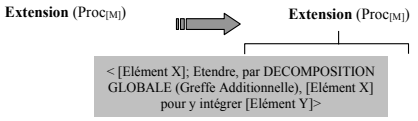
DECOMPOSITION GLOBALE (Greffe Additionnelle) ([Elément X], [Elément Y], [Elément D])	
	**Supprimer** $Proc_{[Elément\ D]}$ **Insérer branche** < [Elément X]; Etendre, par DECOMPOSITION GLOBALE (Greffe Additionnelle), [Elément X] pour y intégrer [Elément Y]> **Insérer arc-plan entre** Extension ($Proc_{[m]}$)**et** < [Elément X]; Etendre, par DECOMPOSITION GLOBALE (Greffe Additionnelle), [Elément X] pour y intégrer [Elément Y]> **Change Graphe-Précédence pour** Extension ($Proc_{[m]}$)

Figure 144 : Principe d'application de la stratégie DECOMPOSITION GLOBALE (Greffe Additionnelle)

18.3.16.2 Exemples d'application de la stratégie DECOMPOSITION GLOBALE (Greffe Additionnelle)

Deux exemples illustrent l'application de cette stratégie à la figure suivante.

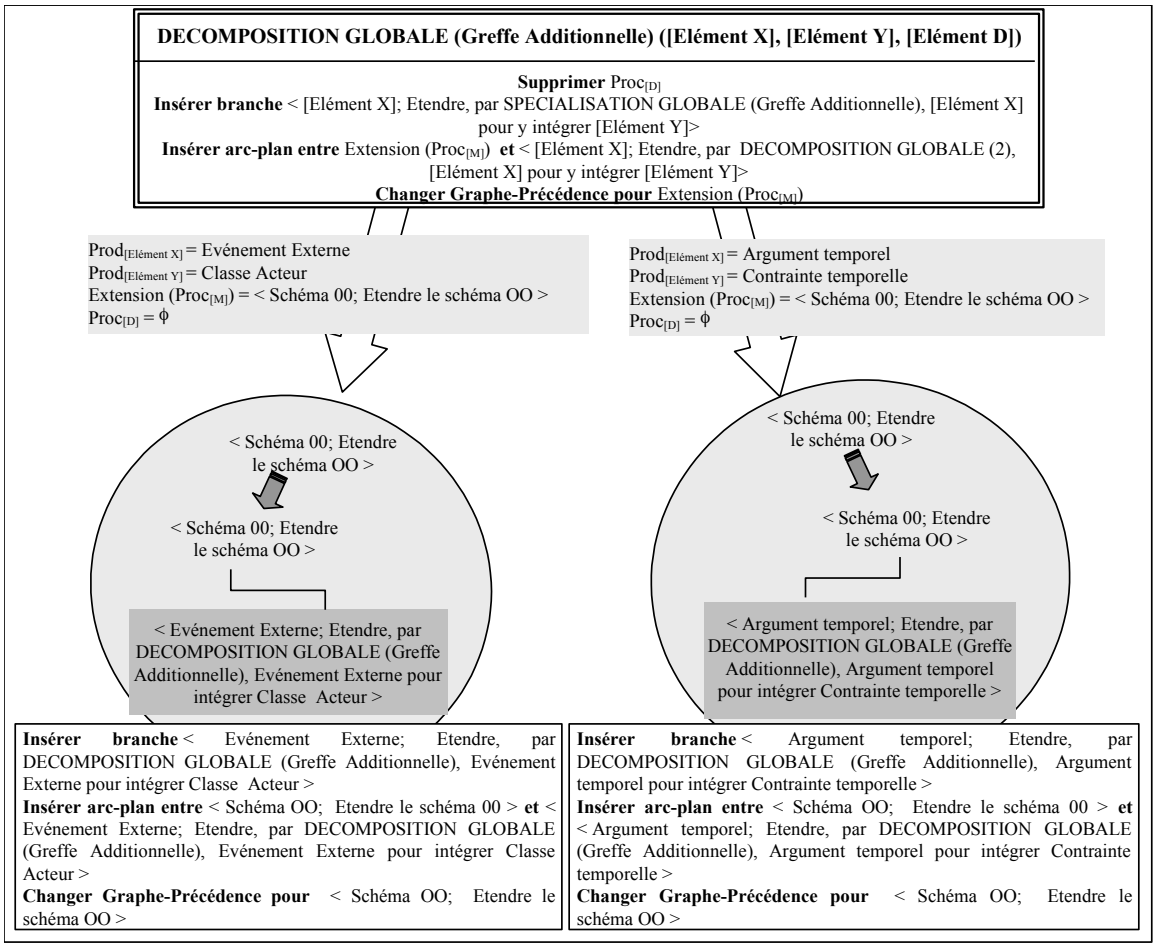


Figure 145: Exemples d'application de la stratégie DECOMPOSITION GLOBALE (Greffe Additionnelle)

Les deux exemples présentés dans cette figure correspondent à l'application de cette stratégie sur deux méthodes ayant déjà été étendues par la stratégie d'extension de la **DÉMARCHE** SEQUENTIELLE GLOBALE et dont le Produit a été étendu par DECOMPOSITION. Dans le premier cas, l'extension ajoute à la démarche représentant les extensions celle de la description du concept de *Classe Acteur* (concept intégré en tant que concept composé de celui d'*Evénement Externe*) alors que, dans le second cas, c'est la démarche de description du concept de *Contrainte Temporelle* qui est greffé dans l'arbre (ce concept ayant été intégré dans le modèle comme composé de celui d'*Argument Temporel*).

18.3.17 DECOMPOSITION LOCALE

On applique cette stratégie lorsque la partie **PRODUIT** a été étendue par la stratégie DECOMPOSITION et que l'ingénieur de méthodes choisit d'étendre la partie **DÉMARCHE** par la stratégie d'EXTENSION LOCALE.

18.3.17.1 Principe d'application de la stratégie DECOMPOSITION LOCALE

Comme pour chacune des stratégies vues précédemment, l'insertion d'un nouvel élément dans la partie **PRODUIT** de la méthode d'origine peut entraîner le fait que l'un des éléments de la méthode devienne inutile. L'extension du **PRODUIT** prend ceci en compte et donne alors la possibilité de

supprimer cet élément. L'extension de la partie **DÉMARCHE** doit donc également proposer de supprimer la construction de cet élément dans la **DÉMARCHE** de la méthode. Ceci s'effectue avec l'opérateur générique ***Supprimer*** *Proc[Elément D]*.

L'application de cette stratégie greffe un processus intermédiaire entre la démarche de description de l'élément que l'on veut étendre - l'Élément X - et son nœud père dans l'arbre. Ce processus intermédiaire permet de mettre en séquence cette démarche de description et une démarche d'extension de ce même élément. La recherche d'une structure générique permettant d'effectuer ce traitement nous a conduit à l'élaboration des opérateurs suivantes.

L'opérateur ***Supprimer arc entre*** *Proc[Elément X]* ***et*** *Parent-de (Proc[Elément X])* permet de couper le lien entre la démarche de description de l'Elément X et son nœud père.

L'opérateur permettant de greffer le processus intermédiaire se formalise de la façon suivante : ***Insérer nœud*** *< [Élément X]; Décrire et étendre [Élément X] >* . Il est ensuite nécessaire de définir si l'arc entre la démarche de description de X et son père est un plan ou un choix, de manière à insérer le processus intermédiaire avec un arc du bon type. Ceci se fait avec la séquence d'opérateurs suivante :

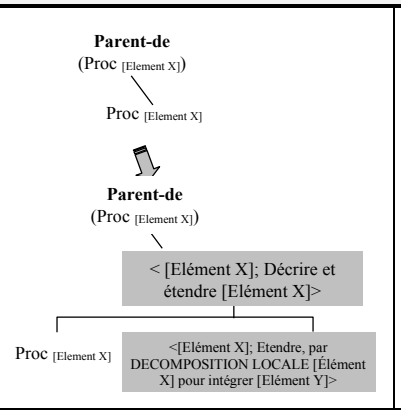
Si *Type-Parent-de (Proc[Elément X]) = PLAN* ***alors***
Insérer arc-plan entre *< [Élément X]; Décrire et Étendre [Élément X] >* ***et***
Parent-de (Proc[Elément X])
sinon
Insérer arc-choix entre *< [Élément X]; Décrire et Étendre [Élément X] >* ***et***
Parent-de (Proc[Elément X])
fsi.

On insère ensuite l'arc de type plan permettant de relier ce processus intermédiaire et la démarche de description de l'élément X avec l'opérateur ***Insérer arc-plan entre*** *< [Élément X]; Décrire et Étendre [Élément X] >* ***et*** *Proc[Elément X]*.

Les opérateurs *{* ***Insérer branche*** *< [Élément X]; Etendre, par DECOMPOSITION LOCALE, [Élément X] pour intégrer [Élément Y] > ;* ***Insérer arc-plan entre*** *< [Élément X]; Décrire et Étendre [Élément X] >* ***et*** *< [Élément X]; Etendre, par DECOMPOSITION LOCALE, [Élément X] pour intégrer [Élément Y] > }* permettent de mettre en séquence les démarches de construction et d'extension.

La dernière étape consiste à construire le graphe de précédence de cette séquence d'extension, ce qui se fait grâce à l'opérateur générique ***Insérer Graphe-Précédence pour*** *< [Élément X]; Décrire et Étendre [Élément X] >*.

DECOMPOSITION LOCALE ([Elément X], [Elément Y], [Elément D])



Supprimer $Proc_{[Elément D]}$
Supprimer arc entre $Proc_{[Elément X]}$ **et Parent-de** ($Proc_{[Elément X]}$)
Insérer nœud < [Elément X]; Décrire et étendre [Elément X] >
Si Type-Parent-de ($Proc_{[Elément X]}$) = PLAN **alors**
Insérer arc-plan entre < [Élément X]; Décrire et Étendre [Élément X] > **et** Parent-de ($Proc_{[Elément X]}$)
sinon
Insérer arc-choix entre < [Élément X]; Décrire et Étendre [Élément X] > **et** Parent-de ($Proc_{[Elément X]}$)
fsi
Insérer arc-plan entre < [Elément X]; Décrire et Etendre [Elément X] > **et** $Proc_{[Elément X]}$
Insérer branche < [Elément X]; Etendre, par DECOMPOSITION LOCALE [Elément X] pour intégrer [Elément Y] >
Insérer arc-plan entre < [Elément X]; Décrire et Etendre [Elément X] > **et** < [Elément X]; Etendre, par DECOMPOSITION LOCALE [Elément X] pour intégrer [Elément Y] >
Insérer Graphe-Précédence pour < [Elément X]; Décrire et Etendre [Elément X] >

Figure 146 : Principe d'application de la stratégie DECOMPOSITION LOCALE

18.3.17.2 Exemples d'application de la stratégie DECOMPOSITION LOCALE

Deux exemples sont présentés dans la figure suivante.

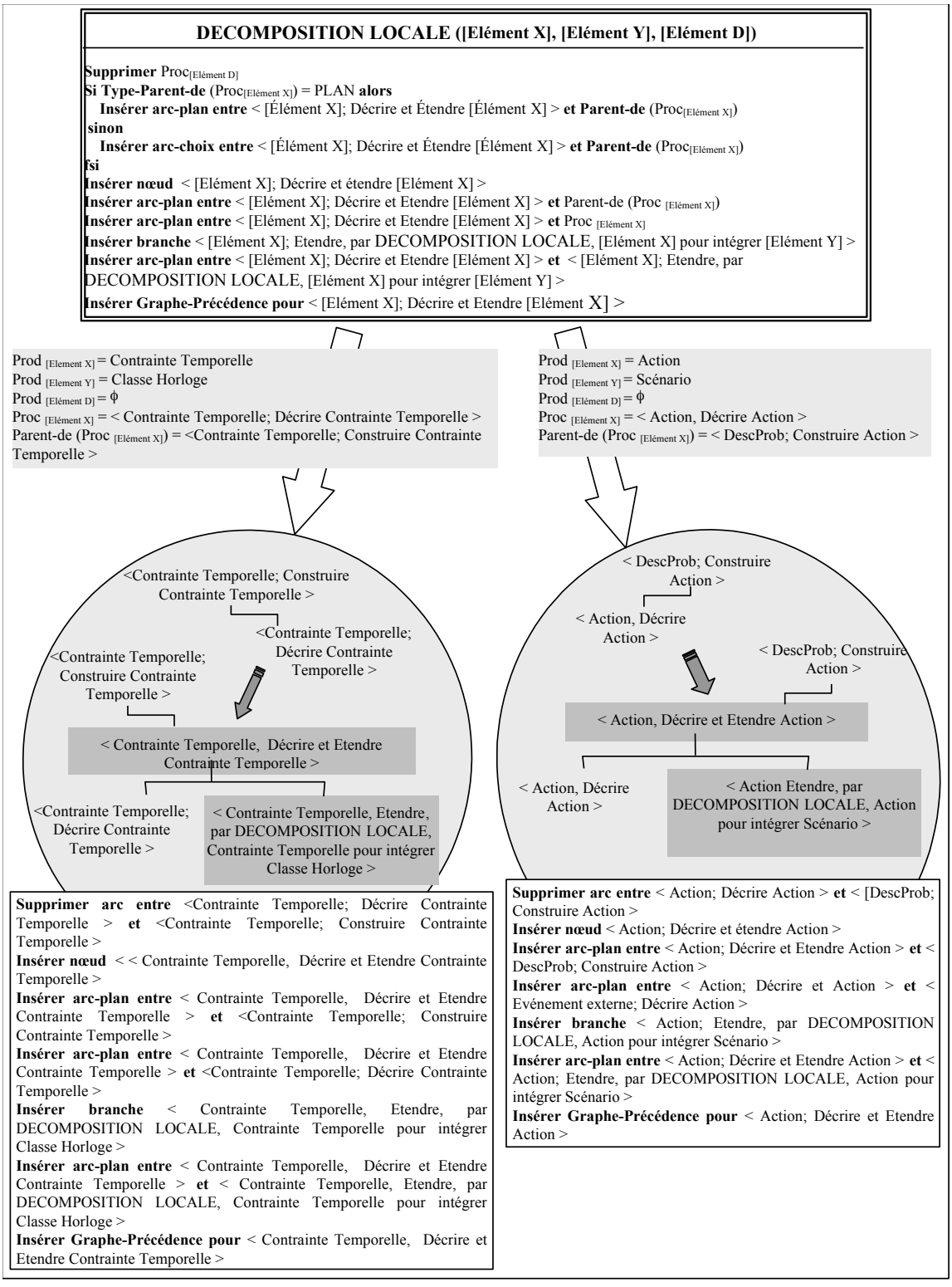


Figure 147: Exemples d'application de la stratégie DECOMPOSITION LOCALE

La figure précédente présentent deux exemples d'extension de méthodes dont le **PRODUIT** a été étendu par DECOMPOSITION et dont l'ingénieur de méthodes souhaite étendre la **DÉMARCHE** par l'application de la stratégie d'EXTENSION SEQUENTIELLE LOCALE. Pour cela, l'extension insère un nœud intermédiaire entre la démarche de description de l'élément que l'on décompose et son nœud père, dans le but de mettre en séquence cette démarche et celle concernant l'extension de cet élément.

Dans le premier cas, c'est le concept de *Contrainte Temporelle* qui est décomposé dans le but d'intégrer celui de *Classe Horloge* alors que dans le second cas, c'est le concept d'*Action* qui est décomposé pour intégrer celui de *Scénario*.

18.3.18 DECOMPOSITION INTEGREE

On applique cette stratégie lorsque la partie **PRODUIT** de la méthode d'origine a été étendue avec la stratégie d'extension de **PRODUIT** DECOMPOSITION et que l'ingénieur de méthodes souhaite étendre la partie **DÉMARCHE** de la méthode avec la stratégie générique d'EXTENSION INTEGREE.

18.3.18.1 Principe d'application de la stratégie DECOMPOSITION INTEGREE

L'application d'une extension peut rendre inutile un élément présent dans la partie **PRODUIT** de la méthode. Il est donc nécessaire, pour la cohérence de la méthode concernée, de supprimer également sa démarche de description dans la partie **DÉMARCHE**. L'opérateur permettant de supprimer cette démarche spécifique est défini de la manière suivante : ***Supprimer*** $Proc_{[Elément\ D]}$ où *[Elément D]* représente l'élément rendu inutile par l'extension et $Proc_{[Elément\ D]}$ sa démarche de description.

On insère ensuite une branche intermédiaire entre la démarche de description de l'[Elément X] et son père, ce qui se fait avec les opérateurs suivants : *{**Supprimer arc entre*** $Proc_{[Elément\ X]}$ ***et Parent-de*** $(Proc_{[Elément\ X]})$ *;* ***Insérer branche*** *< [Élément Y]; Décrire [Élément Y]>*. Il est ensuite nécessaire de définir si l'arc entre la démarche de description de X et son père est un plan ou un choix, de manière à insérer le processus intermédiaire avec un arc du bon type. Ceci se fait avec la séquence d'opérateurs suivante :

Si *Type-Parent-de* $(Proc_{[Elément\ X]})$ *= PLAN* ***alors***
Insérer arc-plan entre *Parent-de* $Proc_{[Élément\ X]}$ ***et*** *< [Elément Y]; Décrire [Elément Y] >*
sinon
Insérer arc-choix entre *Parent-de* $Proc_{[Élément\ X]}$ ***et*** *< [Elément Y]; Décrire [Elément Y] >*
fsi*.*

Puis on rattache la nouvelle démarche à la démarche de description de l'élément X avec l'opérateur ***Insérer arc-plan entre*** *<[Élément Y]; Décrire [Élément Y]>* ***et*** $Proc_{[Elément\ X]}$*>}*. Ces opérateurs permettent de mettre en séquence la démarche de définition du type de l'élément et sa démarche de description.

Finalement, il est nécessaire de définir l'ordre de précédence dans lequel doivent s'exécuter ces deux démarches, ce qui se fait avec *l'opérateur* ***Insérer Graphe-Précédence pour*** *<[Élément Y]; Décrire [Élément Y]>*.

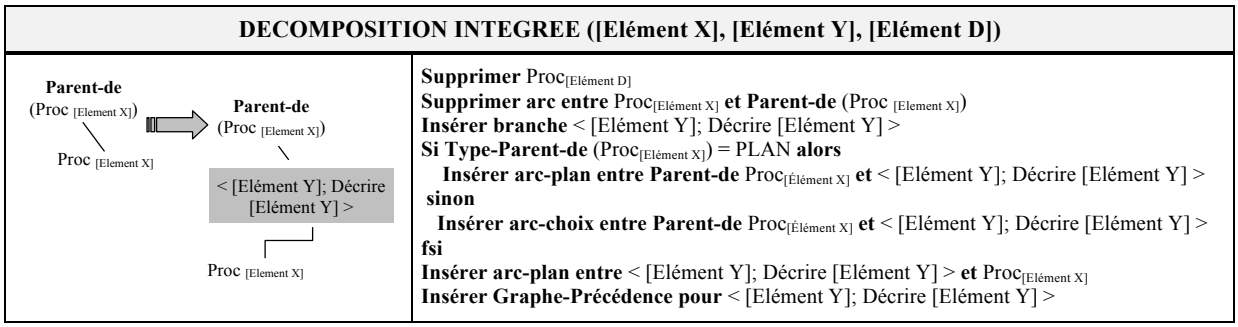


Figure 148 : Principe d'application de la stratégie DECOMPOSITION INTEGREE

18.3.18.2 Exemples d'application de la stratégie DECOMPOSITION INTEGREE

L'application de cette stratégie est illustrée à la figure suivante sur deux exemples d'extension.

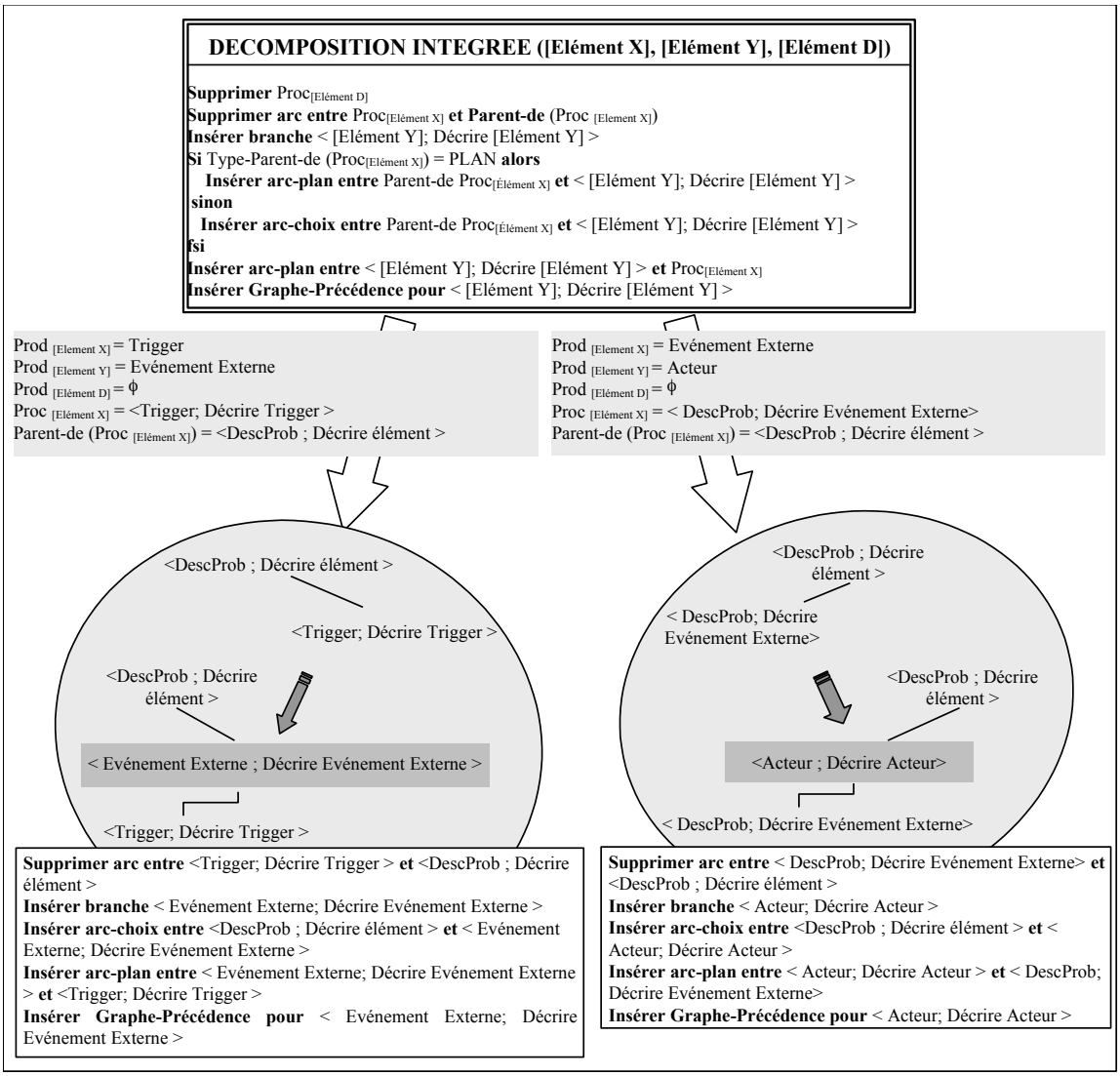


Figure 149: Exemples d'application de la stratégie DECOMPOSITION INTEGREE

La figure précédente illustre l'application de cette stratégie sur deux exemples d'extension par décomposition. En effet, le premier exemple visualise le cas d'une méthode contenant le concept de *Trigger* et que l'on étend pour permettre l'intégration du concept d'*Evénement Externe*. Le concept de *Trigger* pouvant être considéré comme un composé de celui-ci, l'extension s'effectue en greffant une démarche intermédiaire entre la description du *Trigger* et son père dans l'arbre de processus. De la même manière, le deuxième exemple illustre le cas d'une méthode possédant le concept d'*Evénement Externe* et que l'on étend pour intégrer le concept d'*Acteur*, concept composé de celui d'*Evénement Externe*.

18.3.19 REFERENCE GLOBALE (1ere Greffe)

On applique cette stratégie lorsque la partie **PRODUIT** a été étendue par la stratégie REFERENCE et que l'ingénieur de méthodes choisit d'étendre la partie **DÉMARCHE** par la stratégie d'EXTENSION SEQUENTIELLE GLOBALE (1ère Greffe).

18.3.19.1 Principe d'application de la stratégie REFERENCE GLOBALE (1ère Greffe)

Comme pour les stratégies précédentes, la première opération à effectuer sur la **DÉMARCHE** de la méthode est de supprimer la construction de tout élément rendu inutile par l'extension de la partie **PRODUIT**. Ceci se fait avec l'opérateur ***Supprimer*** $Proc_{[Elément\ D]}$.

Comme il n'y a pas eu de greffe préalable suivant cette stratégie particulière, il va falloir créer une nouvelle branche de l'arbre de processus pour prendre en compte toute nouvelle extension suivant cette stratégie. Ceci s'effectue en ajoutant une nouvelle racine à l'arbre pour mettre en séquence la construction orientée objet initiale puis les modifications d'extension. Ces modifications s'effectuent grâce aux opérateurs suivants : {***Insérer nœud*** < *Description du Problème; Construire et étendre le schéma 00* > ; ***Insérer arc-plan entre*** < *Description du Problème; Construire et étendre le schéma OO* > ***et*** *Racine* ($Proc_{[M]}$) ; ***Insérer branche*** < *Schéma OO; Étendre le schéma OO* >}, où la fonction *Racine* ($Proc_{[M]}$) correspond au processus présent à la racine de l'arbre de processus et qui représente la branche permettant la construction du schéma objet avant toute extension.

Insérer un nouveau nœud comme racine de l'arbre de processus insère également toutes ses démarches sous-jacentes. Cette stratégie insère donc également un nœud fils représentant les extensions (appelé <*Schéma OO; Etendre le schéma OO* >) ne contenant après l'exécution de cette stratégie qu'une seule démarche étant < *[Élément X]; Etendre, par REFERENCE GLOBALE (1ère Greffe), [Élément X] pour intégrer [Élément Y]* >.

La dernière étape consiste à construire le graphe de précédence de cette séquence, ce qui se fait grâce à l'opérateur ***Insérer Graphe-Précédence pour*** < *Description du Problème; Construire et étendre le schéma OO* >.

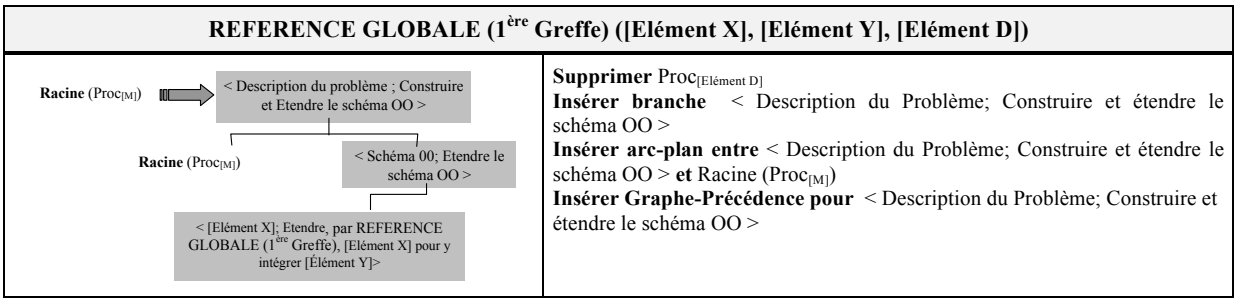


Figure 150 : Principe d'application de la stratégie REFERENCE GLOBALE (1ère Greffe)

18.3.19.2 Exemples d'application de la stratégie REFERENCE GLOBALE (1ère Greffe)

L'application de cette stratégie est illustrée par les deux exemples de la figure suivante.

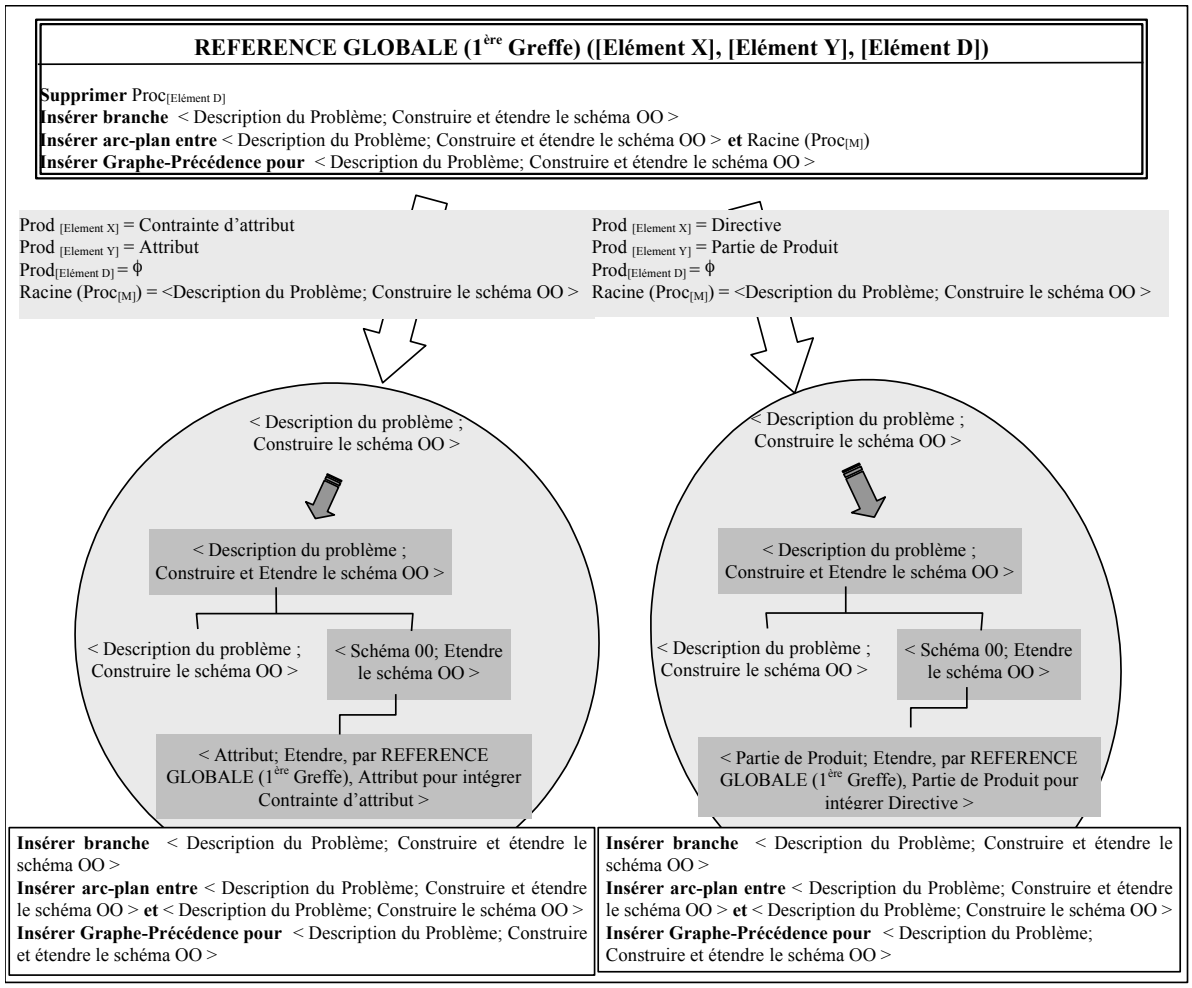


Figure 151: Exemples d'application de la stratégie REFERENCE GLOBALE (1ère Greffe)

La figure précédente illustre l'application de cette stratégie sur deux exemples. Comme il a été expliqué précédemment, cette stratégie s'applique lorsque la partie **PRODUIT** a été étendue par REFERENCE et que l'ingénieur de méthodes souhaite étendre la partie **DÉMARCHE** avec une EXTENSION SEQUENTIELLE GLOBALE. Il est de plus nécessaire d'avoir une méthode dont la

DÉMARCHE n'a jamais été étendue avec cette stratégie spécifique. L'extension de la méthode s'organise donc autour de la greffe d'une nouvelle racine dans l'arbre, ce qui permet de mettre en séquence la démarche de construction du schéma OO et son extension. On ajoute ensuite la démarche d'extension qui ne comportera, pour l'instant, qu'une seule démarche, celle concernant l'extension qui nous intéresse. Dans le premier cas, il s'agit de l'extension du concept d'*Attribut* pour l'associer à un nouveau concept permettant de définir des *Contrainte(s) d'attribut*. Dans le second cas, on ajoute au concept *Partie de Produit* une référence avec un nouveau concept appelé *Directive*.

18.3.20 REFERENCE GLOBALE (Greffe Additionnelle)

Cette stratégie s'applique lorsque l'ingénieur de méthodes a déjà préalablement exécuté un patron d'extension de la **DÉMARCHE** avec la stratégie d'EXTENSION SEQUENTIELLE GLOBALE, que la partie **PRODUIT** a été étendue grâce à la stratégie REFERENCE et qu'il souhaite de nouveau exécuter cette stratégie spécifique d'extension de la **DÉMARCHE**.

18.3.20.1 Principe d'application de la stratégie REFERENCE GLOBALE (Greffe Additionnelle)

De même que pour toutes les stratégies décrites jusqu'ici, le premier opérateur de transformation d'une stratégie d'extension du **PRODUIT** étant une suppression d'un possible élément de **PRODUIT** rendu inutile par l'extension de la méthode, il faut, lors de l'extension de la partie **DÉMARCHE**, supprimer toute trace de cet élément. Il est donc nécessaire de supprimer la **DÉMARCHE** de construction de cet élément dans l'arbre de processus. Cet opérateur s'écrit de la manière suivante : ***Supprimer*** $Proc_{[Elément\ D]}$, où *[Elément D]* représente l'élément caduc et $Proc_{[Elément\ D]}$ son processus de construction.

La partie **DÉMARCHE** ayant déjà été étendue avec cette stratégie, il existe déjà une branche de l'arbre de processus qui concerne les extensions séquentielles globales. Il faut donc greffer une nouvelle démarche dans la séquence pour prendre en compte celle qui nous intéresse. Cet ajout ce fait grâce à l'opérateur ***Insérer arc-plan entre*** *Extension (Proc[M])* ***et*** *< [Élément X] ; Etendre, par REFERENCE GLOBALE (Greffe Additionnelle), [Élément X] pour intégrer [Élément Y]>* où *[Élément X]* représente l'élément qui était déjà présent dans la méthode d'origine et *[Élément Y]* l'élément qui a été intégré grâce à un patron d'extension du **PRODUIT**.

Il est ensuite nécessaire de réorganiser le graphe de transition d'états de la séquence de ce noeud car certaines extensions peuvent avoir des heuristiques de précédence entre elles. Ceci se fait avec l'opération ***Changer Graphe-Précédence pour*** *Extension (Proc[M])*.

REFERENCE GLOBALE (Greffe Additionnelle) ([Elément X], [Elément Y], [Elément D])	
Extension ($Proc_{[M]}$) ⇒ **Extension** ($Proc_{[M]}$) < [Elément X]; Etendre, par REFERENCE GLOBALE (Greffe Additionnelle), [Elément X] pour y intégrer [Elément Y]>	**Supprimer** $Proc_{[Elément\ D]}$ **Insérer branche** < [Elément X]; Etendre, par REFERENCE GLOBALE (Greffe Additionnelle), [Elément X] pour y intégrer [Elément Y]> **Insérer arc-plan entre** Extension ($Proc_{[m]}$)**et** < [Elément X]; Etendre, par REFERENCE GLOBALE (Greffe Additionnelle), [Elément X] pour y intégrer [Elément Y]> **Change Graphe-Précédence pour** Extension ($Proc_{[m]}$)

Figure 152 : Principe d'application de la stratégie REFERENCE GLOBALE (Greffe Additionnelle)

18.3.20.2 Exemples d'application de la stratégie REFERENCE GLOBALE (Greffe Additionnelle)

L'application de cette stratégie est illustrée à la figure suivante.

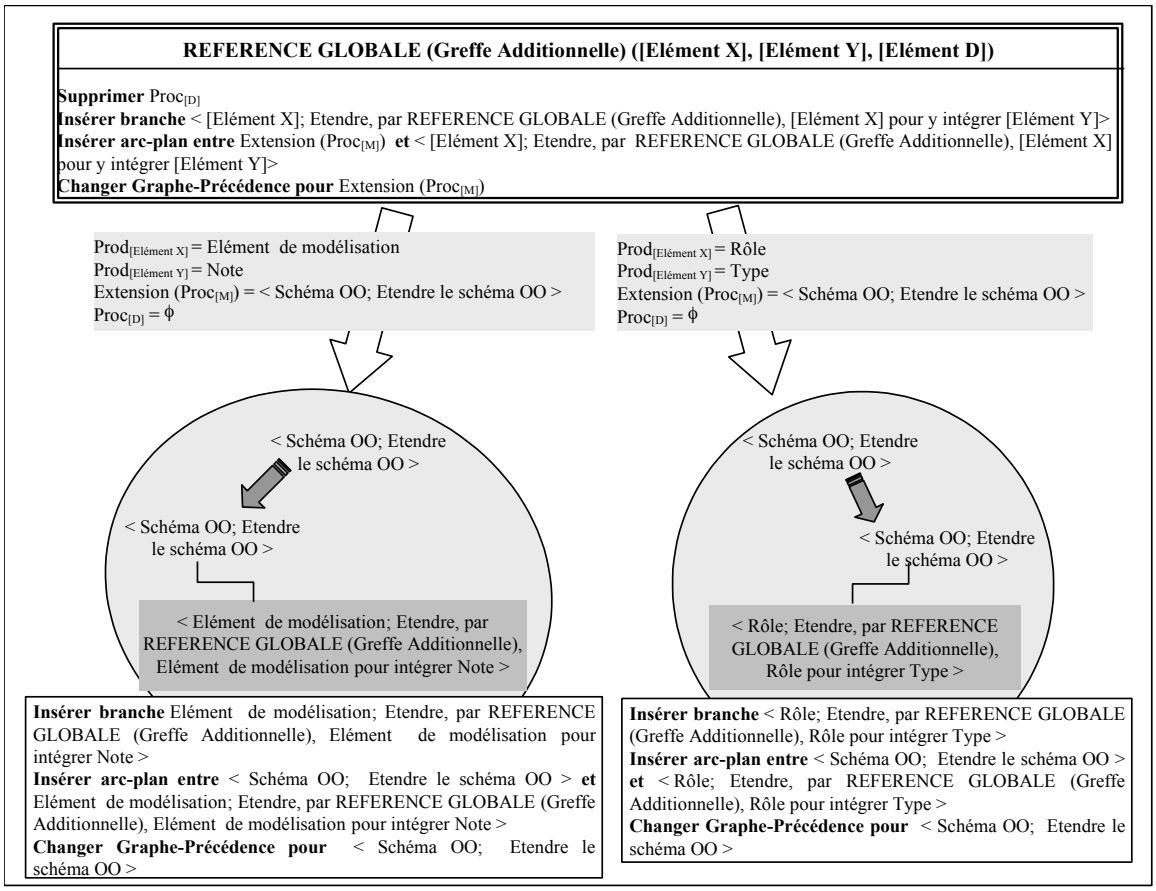


Figure 153: Exemples d'application de la stratégie REFERENCE GLOBALE (Greffe Additionnelle)

Cette stratégie s'applique lorsque l'élément qui nous intéresse a été intégré dans la partie **PRODUIT** de la méthode par REFERENCE, que la partie **DÉMARCHE** a déjà été étendue de façon SEQUENTIELLE GLOBALE et que l'ingénieur de méthodes décide d'intégrer la démarche de description de cet élément avec cette même stratégie. Dans le premier cas, on greffe au nœud concernant les extensions la démarche permettant d'intégrer la description du concept de *Note* comme concept référencé par celui d'*Elément*. Dans le second cas, c'est la démarche de description du concept de *Type* qui est intégré avec ce concept référençant celui de *Rôle*.

18.3.21 REFERENCE LOCALE

On applique cette stratégie lorsque la partie **PRODUIT** a été étendue par la stratégie REFERENCE et que l'ingénieur de méthodes choisit d'étendre la partie **DÉMARCHE** par la stratégie d'EXTENSION SEQUENTIELLE LOCALE.

18.3.21.1 Principe d'application de la stratégie REFERENCE LOCALE

De la même façon que pour toutes les autres stratégies expliquées précédemment, la première modification effectuée par une stratégie d'extension est une suppression d'un possible élément de **PRODUIT** rendu inutile par l'extension de la méthode. Lors de l'extension de la partie **DÉMARCHE**, il est nécessaire de supprimer également toute trace de cet élément dans l'arbre de processus (suppression de la **DÉMARCHE** de construction de cet élément). Cet opérateur peut s'écrire de la manière suivante : ***Supprimer*** $Proc_{[Elément\ D]}$, où *[Elément D]* représente l'élément inutile et $Proc_{[Elément\ D]}$ son processus de construction.

Appliquer cette stratégie greffe un processus intermédiaire entre le processus de description de l'élément que l'on veut étendre - $Proc_{[Elément\ X]}$ - et son père dans l'arbre de processus - *Parent-de* ($Proc_{[Elément\ X]}$). Ce processus intermédiaire permettra de mettre en séquence la **DÉMARCHE** de description et une nouvelle **DÉMARCHE** d'extension de ce même élément. La recherche d'une structure générique permettant d'effectuer ce traitement nous a conduit à l'élaboration des opérateurs suivants.

L'opérateur ***Supprimer arc entre*** $Proc_{[Elément\ X]}$ ***et Parent-de*** ($Proc_{[Elément\ X]}$) permet de couper le lien entre la démarche de description de l'Elément X et son nœud père.

L'opérateur permettant de greffer le processus intermédiaire se formalise de la façon suivante : ***Insérer nœud*** *< [Élément X]; Décrire et étendre [Élément X] >* . Il est ensuite nécessaire de définir si l'arc entre la démarche de description de X et son père est un plan ou un choix, de manière à insérer le processus intermédiaire avec un arc du bon type. Ceci se fait avec la séquence d'opérateurs suivante :

Si *Type-Parent-de* ($Proc_{[Elément\ X]}$) *= PLAN* ***alors***
Insérer arc-plan entre *< [Élément X]; Décrire et Étendre [Élément X] >* ***et***
Parent-de ($Proc_{[Elément\ X]}$)
sinon
Insérer arc-choix entre *< [Élément X]; Décrire et Étendre [Élément X] >* ***et***
Parent-de ($Proc_{[Elément\ X]}$)
fsi.

On insère ensuite l'arc de type plan permettant de relier ce processus intermédiaire et la démarche de description de l'élément X avec l'opérateur ***Insérer arc-plan entre*** *< [Élément X]; Décrire et Étendre [Élément X] >* ***et*** $Proc_{[Elément\ X]}$.

Les opérateurs *{* ***Insérer branche*** *< [Élément X]; Etendre, par REFERENCE LOCALE, [Élément X] pour intégrer [Élément Y] > ;* ***Insérer arc-plan entre*** *< [Élément X]; Décrire et Étendre [Élément X] >* ***et*** *< [Élément X];]; Etendre, par REFERENCE LOCALE, [Élément X] pour intégrer [Élément Y] >}* permettent de mettre en séquence les démarches de construction et d'extension.

Enfin, la dernière étape consiste à construire le graphe de précédence de cette séquence d'extension, ce qui se fait grâce à l'opérateur ***Insérer Graphe-Précédence pour*** *< [Élément X]; Décrire et Étendre [Élément X] >*.

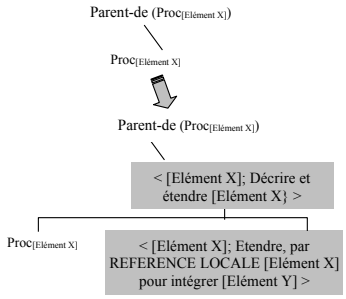
REFERENCE LOCALE ([Elément X], [Elément Y], [Elément D])	
	**Supprimer** Proc[Elément D] **Supprimer arc entre** Proc[Elément X] **et** Parent-de (Proc[Elément X]) **Insérer nœud** < [Elément X]; Décrire et étendre [Elément X] > **Si** Type-Parent-de (Proc[Elément X]) = PLAN **alors** **Insérer arc-plan entre** < [Élément X]; Décrire et Étendre [Élément X] > **et** Parent-de (Proc[Elément X]) **sinon** **Insérer arc-choix entre** < [Élément X]; Décrire et Étendre [Élément X] > **et** Parent-de (Proc[Elément X]) **fsi** **Insérer arc-plan entre** < [Elément X]; Décrire et Etendre [Elément X] > **et** Proc[Elément X] **Insérer branche** < [Elément X]; Etendre, par REFERENCE LOCALE [Elément X] pour intégrer [Elément Y] > **Insérer arc-plan entre** < [Elément X]; Décrire et Etendre [Elément X] > **et** < [Elément X]; Etendre, par REFERENCE LOCALE [Elément X] pour intégrer [Elément Y] > **Insérer Graphe-Précédence pour** < [Elément X]; Décrire et Etendre [Elément X] >

Figure 154 : Principe d'application de la stratégie REFERENCE LOCALE

18.3.21.2 Exemples d'application de la stratégie REFERENCE LOCALE

La figure suivante illustre l'application de cette stratégie sur deux exemples d'extension de méthodes.

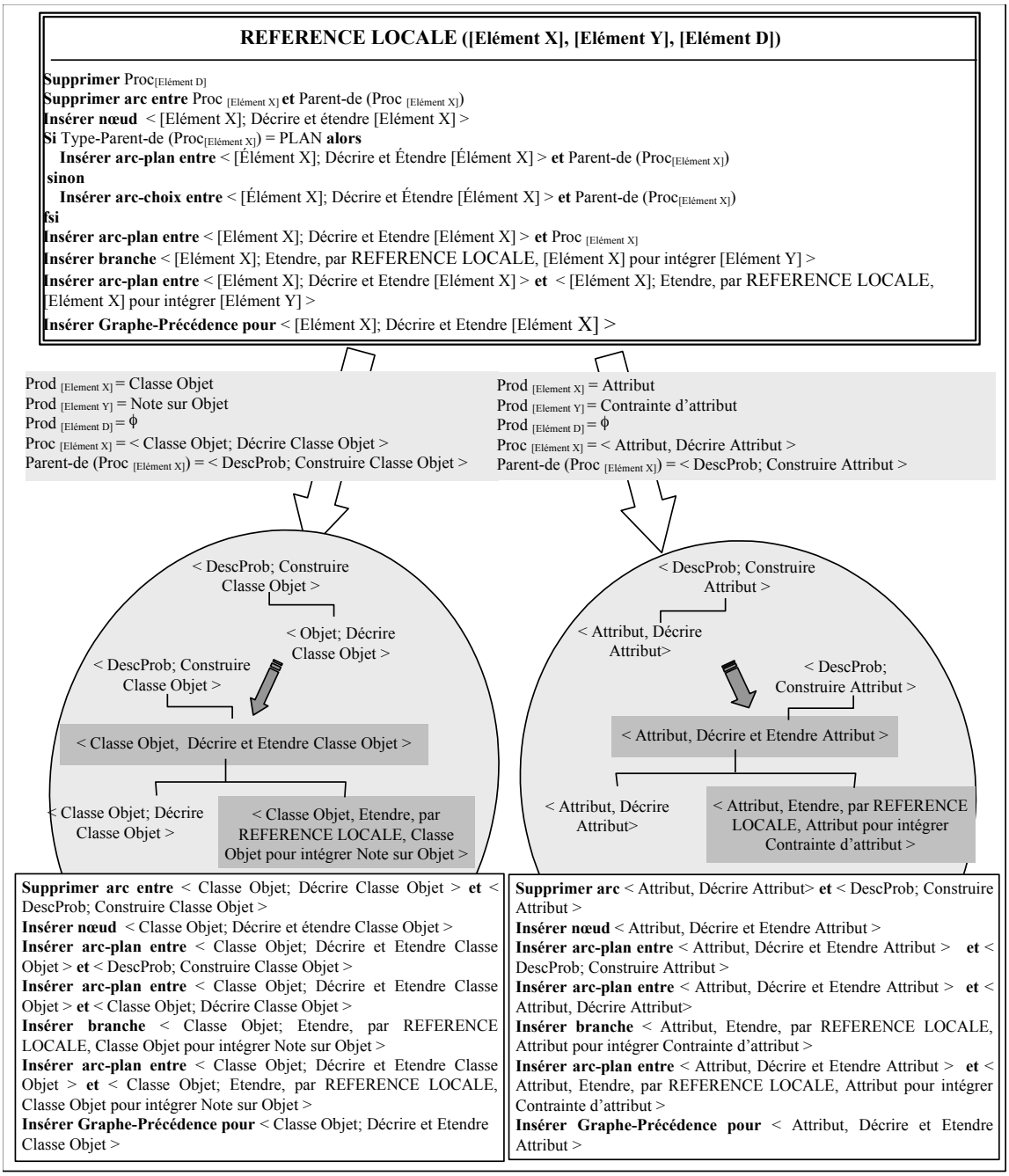


Figure 155: Exemples d'application de la stratégie REFERENCE LOCALE

Ces exemples visualisent l'application de cette stratégie sur deux parties de **DÉMARCHE** d'une méthode. Le premier exemple illustre l'extension d'une méthode dans le but de lui intégrer la démarche de construction du concept de *Note sur Objet*, concept en référence avec celui de *Classe Objet*. L'extension ajoute donc un nœud intermédiaire entre la démarche de description de *Classe Objet* et son nœud père, ce qui permet de mettre en séquence cette démarche et celle concernant l'extension de *Classe Objet* avec *Note sur Objet*. De même, dans le second exemple, c'est la démarche de description du concept de *Contrainte d'Attribut* qui est intégrée à la suite de celle du concept d'*Attribut*.

18.3.22 REFERENCE INTEGREE

On applique cette stratégie lorsque la partie **PRODUIT** de la méthode d'origine a été étendue avec la stratégie d'extension de **PRODUIT** REFERENCE et que l'ingénieur de méthodes souhaite étendre la partie **DÉMARCHE** de la méthode avec la stratégie générique d'EXTENSION INTEGREE.

18.3.22.1 Principe d'application de la stratégie REFERENCE INTEGREE

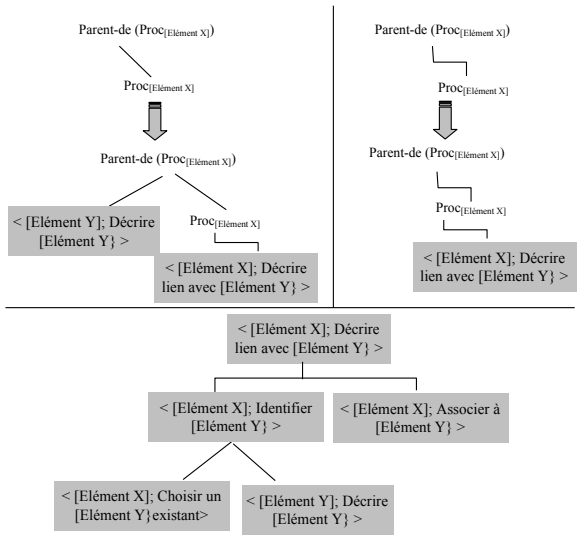
La première opération effectuée par une stratégie d'extension est une suppression d'un élément de **PRODUIT** rendu inutile par l'extension de la méthode, si celui-ci existe. Lors de l'extension de la partie **DÉMARCHE**, il est nécessaire de supprimer également la **DÉMARCHE** de construction de cet élément. Cet opérateur se formalise de la manière suivante : ***Supprimer*** $Proc_{[Elément\ D]}$, où *[Elément D]* représente l'élément inutile et $Proc_{[Elément\ D]}$ son processus de construction.

Appliquer cette stratégie greffe un nouveau nœud dans la séquence de description de l'[Elément X]. Ceci se fait avec les opérateurs {***Insérer nœud*** *<[Elément X] ; Décrire lien avec [Elément Y]>* ; ***Insérer arc-plan entre*** $Proc_{[Elément\ X]}$ ***et*** *<[Elément X] ; Décrire lien avec [Elément Y]>* }. Ces opérateurs sont suivis de celui permettant de réorganiser le graphe de précédence de cette séquence : ***Changer GraphePrécédence pour*** $Proc_{[Elément\ X]}$.

Il est ensuite nécessaire de définir si l'arc entre la démarche de description de X et son père est un choix. En effet, si c'est le cas, il y a un opérateur supplémentaire a exécuter permettant d'insérer un arc entre la démarche de description de l'[Elément Y] et le nœud père de la démarche de description de l'[Elément X]. La séquence suivante permet d'effectuer cette différenciation:

Si *type-parent-de* ($Proc_{[Elément\ X]}$) *= CHOIX* ***alors***
  Insérer arc-choix entre *Parent-de* ($Proc_{[Elément\ X]}$) ***et*** *<[Elément Y] ; Décrire [Elément Y]>*
fsi.

REFERENCE INTEGREE ([Elément X], [Elément Y], [Elément D])



Supprimer Proc[Elément D]
Insérer branche <[Elément X] ; Décrire lien avec [Elément Y]>
Insérer arc-plan entre Proc[Elément X] **et** <[Elément X] ; Décrire lien avec [Elément Y]>
Changer GraphePrécédence pour Proc[Elément X]
Si type-parent-de (Proc[Elément X]) = CHOIX **alors**
  Insérer arc-choix entre Parent-de (Proc[Elément X]) **et** <[Elément Y] ; Décrire [Elément Y]>
fsi

Figure 156 : Principe d'application de la stratégie REFERENCE INTEGREE

18.3.22.2 Exemples d'application de la stratégie REFERENCE INTEGREE

L'application de cette stratégie est illustrée à la figure suivante avec deux exemples d'extension.

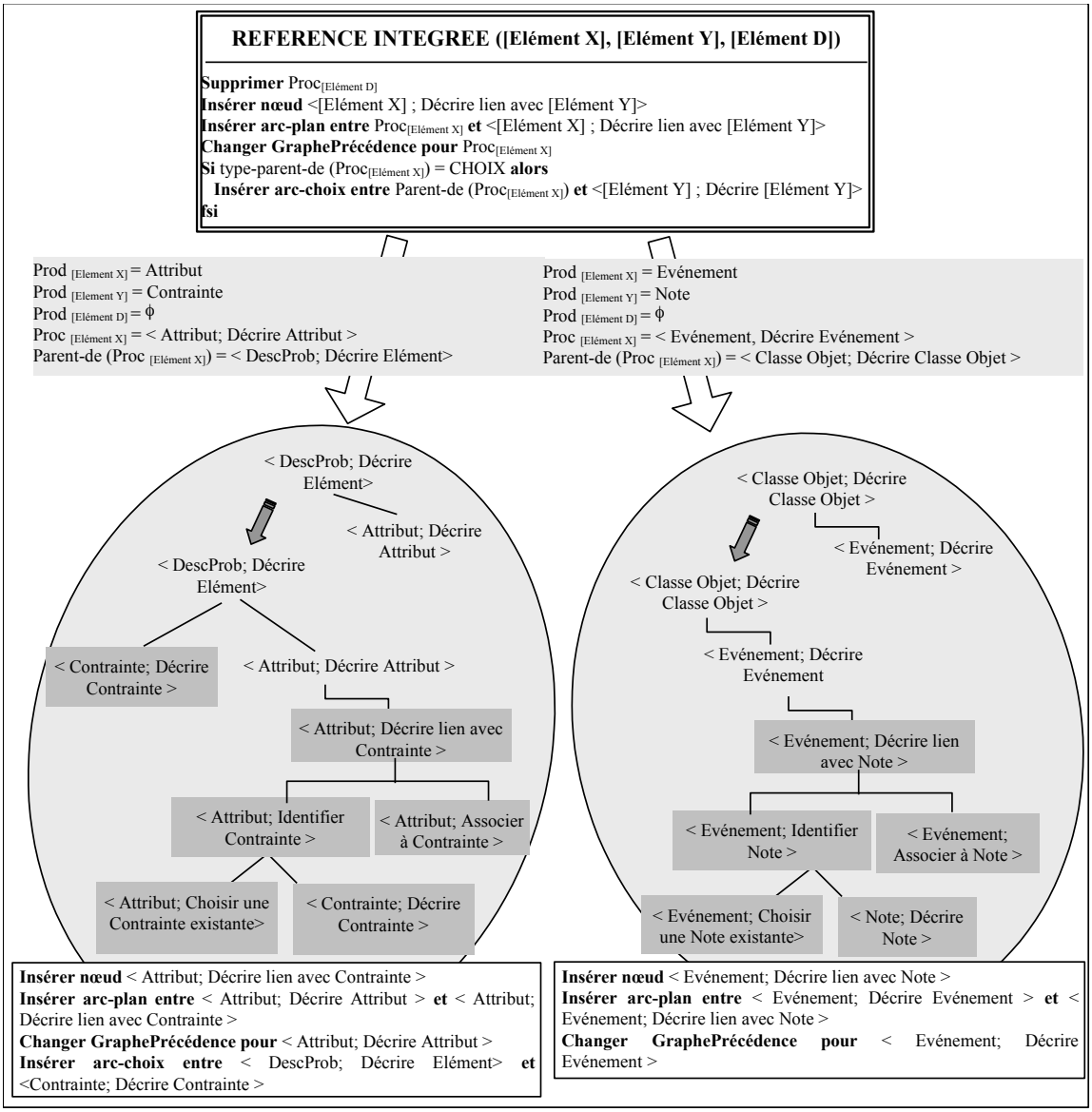


Figure 157: Exemples d'application de la stratégie REFERENCE INTEGREE

Les exemples illustrés à la figure précédente montre l'application de cette stratégie d'extension. Le premier exemple illustre le cas d'une méthode possédant le concept d'*Attribut* et que l'ingénieur de méthodes a étendu avec le concept de *Contrainte*, ceux-ci étant reliés par le biais d'un lien de référence. L'extension de la **DÉMARCHE** de la méthode de façon intégrée permet d'ajouter la démarche d'association de ces concepts à celle de la description d'*Attribut*. De plus, l'extension permet également de greffer la démarche de description de *Contrainte* comme alternative à celle d'*Attribut*. Contrairement au deuxième exemple où le nœud père de la démarche de description de l'élément que l'on étend (ici *Evénement*) est un contexte de type PLAN, ce qui ne permet pas d'effectuer cette greffe de l'alternative. L'extension se restreint donc à la spécification de l'association entre les concepts d'*Evénement* et de *Note* dans la séquence de description de *Evénement*.

18.3.23 REMPLACEMENT INTEGRE

On applique cette stratégie lorsque la partie **PRODUIT** de la méthode d'origine a été étendue avec la stratégie d'extension de **PRODUIT** REMPLACEMENT et que l'ingénieur de méthodes souhaite étendre la partie **DÉMARCHE** de la méthode avec la stratégie générique d'EXTENSION INTEGREE.

18.3.23.1 Principe d'application de la stratégie REMPLACEMENT INTEGRE

Contrairement aux autres stratégies décrites dans cette section, la stratégie d'extension de **PRODUIT** par REMPLACEMENT n'induit pas une possibilité d'élément rendu caduc mais entraîne obligatoirement cet inutilité d'un ancien élément. En conséquence, il n'est pas donné à l'ingénieur de méthodes la possibilité de supprimer cet ancien élément et l'application de cette stratégie a comme automatique conséquence cette suppression. Au niveau de l'extension de la **DÉMARCHE** en suivant cette stratégie d'EXTENSION INTEGREE, il suffira de renommer le nœud concernant la description de cet élément pour permettre de commencer le remplacement. L'opérateur permettant de renommer un sommet est le suivant ***Renommer*** $Proc_{[Elément\ X]}$ ***avec*** *<[Elément Y], Décrire [Elément Y]>* où *[Elément X]* représente l'élément que l'on souhaite remplacer, $Proc_{[Elément\ X]}$ son processus de description et *[Elément Y]* l'élément que l'on souhaite intégrer à sa place.

L'application de cette stratégie va également remplacer la démarche de description de la structure de l'[Elément X] par la description de la structure de l'[Elément Y] mais va conserver la description des liens que l'[Elément X] avait avec le reste du modèle (démarches contenues dans l'[Ensemble W]). La recherche d'une structure générique permettant d'effectuer ce traitement nous a conduit à l'élaboration des opérateurs suivants *{**Supprimer** <[Elément X] ; Décrire structure [Elément X]> ; **Insérer** <[Elément Y] ; Décrire structure [Elément Y]> ; **Insérer Arc-plan entre** <[Elément Y] ; Décrire [Elément Y]> **et** <[Elément Y] ; Décrire structure [Elément Y]>}*

Enfin, la dernière étape consiste à réorganiser le graphe de précédence de cette séquence d'extension, ce qui se fait grâce à l'opérateur ***Insérer Graphe-Précédence pour*** *< [Élément Y]; Décrire [Élément Y] >*.

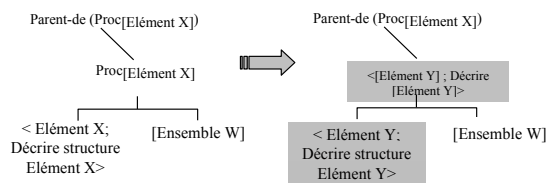
REMPLACEMENT INTEGRE ([Elément X], [Elément Y])	
	**Renommer** Proc[Elément X] **avec** <[Elément Y] ; Décrire [Elément Y]> **Supprimer** <[Elément X] ; Décrire structure [Elément X]> **Insérer** <[Elément Y] ; Décrire structure [Elément Y]> **Insérer Arc-plan entre** <[Elément Y] ; Décrire [Elément Y]> **et** <[Elément Y] ; Décrire structure [Elément Y]> **Insérer Graphe-Précédence pour** < [Élément Y]; Décrire [Élément Y] >

Figure 158 : Principe d'application de la stratégie REMPLACEMENT INTEGRE

18.3.23.2 Exemples d'application de la stratégie REMPLACEMENT INTEGRE

La figure suivante illustre l'application de cette stratégie sur deux exemples d'extension.

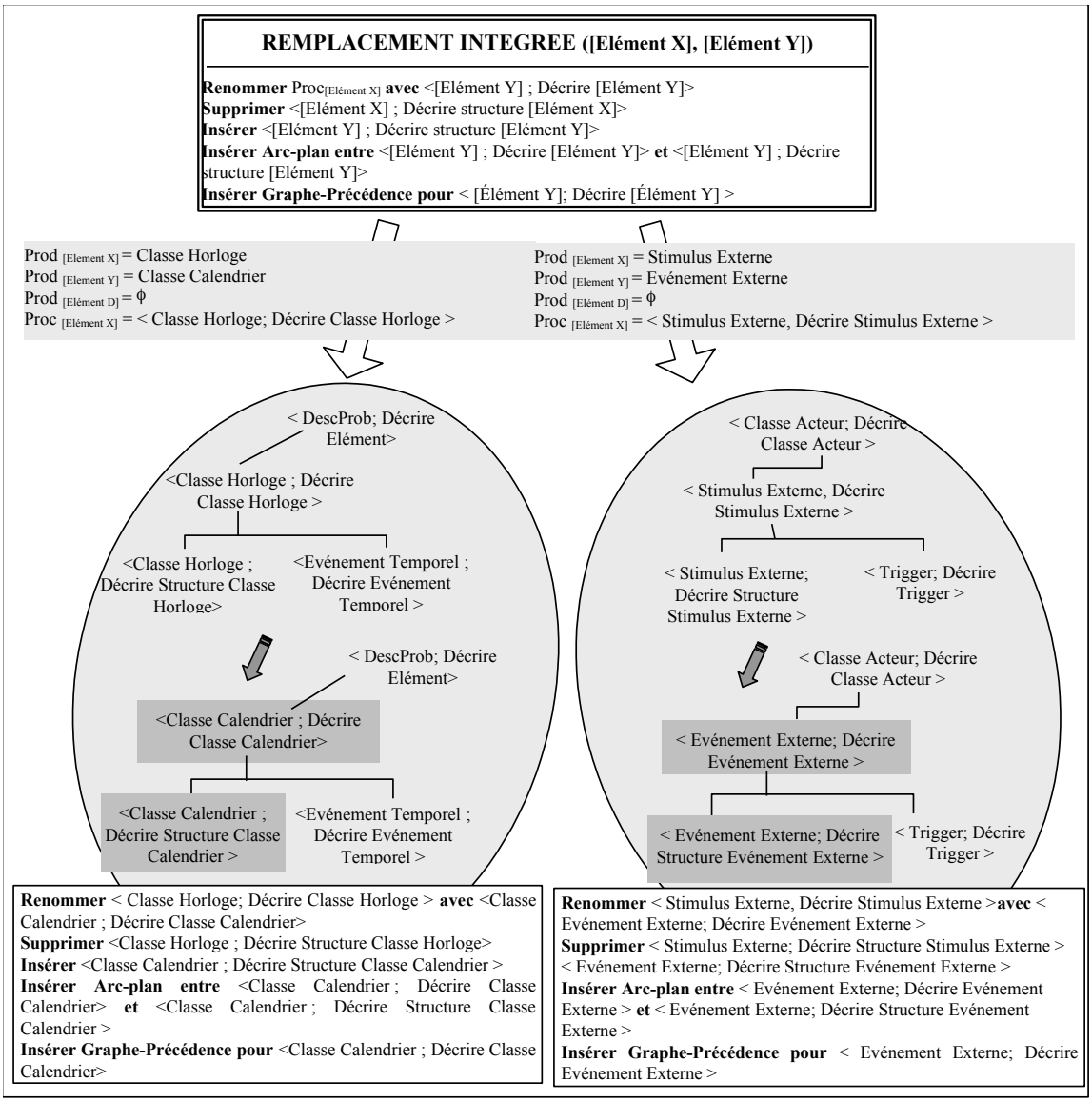


Figure 159: Exemples d'application de la stratégie REMPLACEMENT INTEGREE

Le premier exemple visualisé à cette figure montre l'application de cette stratégie sur une méthode dont le **PRODUIT** a été étendu pour intégrer le concept de *Classe Calendrier* par remplacement de celui de *Classe Horloge*. L'extension de la **DÉMARCHE** permet de conserver les démarches de description des liens concernant l'ancien concept et de ne remplacer que sa structure par celle de *Classe Calendrier*. De même, dans le second exemple, c'est la démarche de description du concept de *Stimulus Externe* qui est modifiée pour prendre ne compte la description de la structure du nouveau concept d'Evénement Externe.

Chapitre VII : Exemple d'extension de méthode au domaine temporel

Introduction

Nous allons illustrer les propos tenus dans ce mémoire par l'exemple de l'extension de la méthode orientée objet O* aux applications temporelles. La Figure 160 illustre le processus de construction et d'utilisation des patrons d'extensions permettant ces modifications.

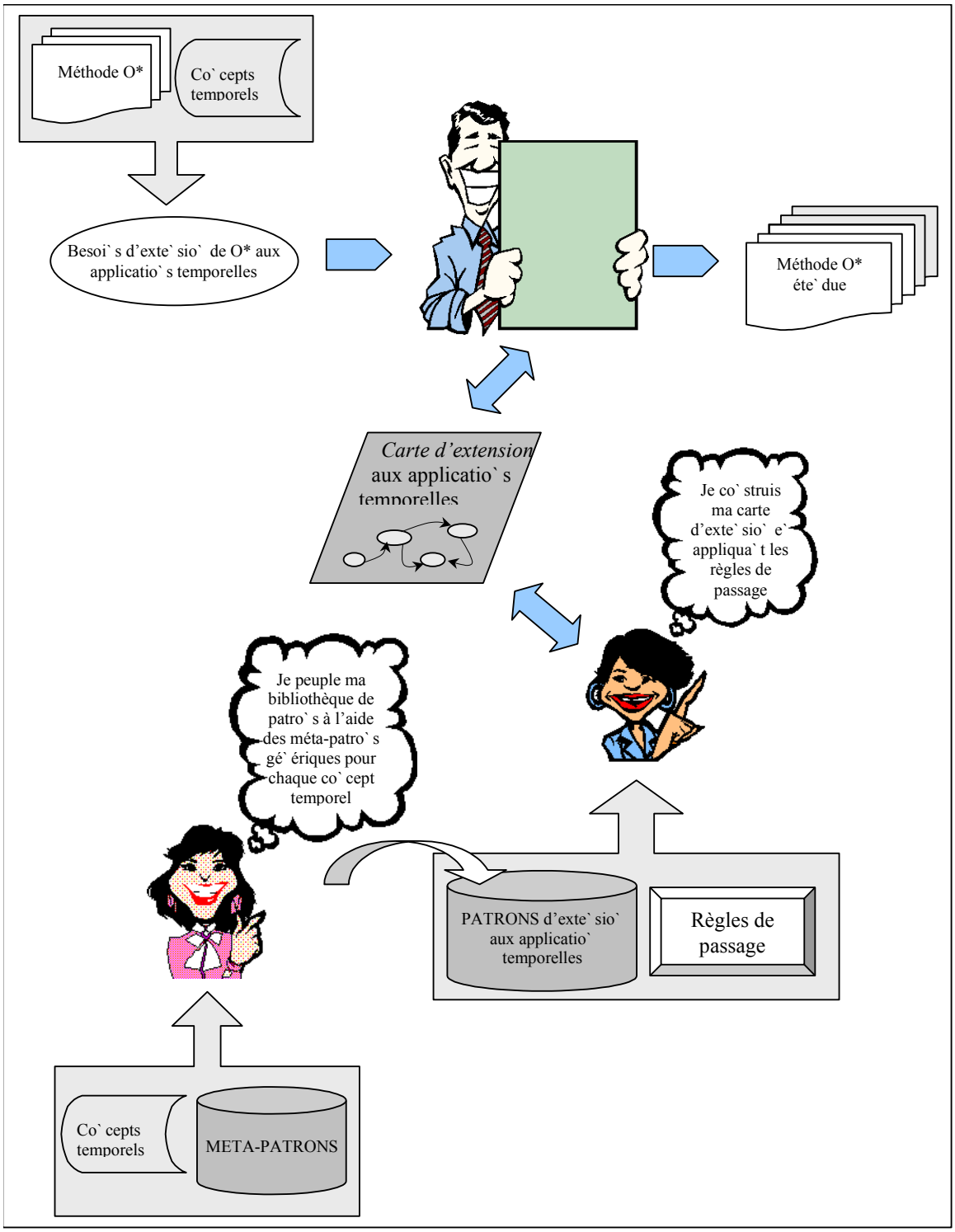


Figure 160: Processus d'extension de la méthode O* aux applications temporelles

Nous pouvons distinguer trois étapes spécifiques sur cette figure : la construction des patrons d'extension pour les applications temporelles, la construction de la carte d'extension associée et l'exécution de cette carte pour la méthode O*.

Première étape : Dans un premier temps, l'ingénieur de méthode définit les concepts temporels présents dans les applications temporelles. Puis, dans un deuxième temps, il applique chaque méta-patron d'extension de **PRODUIT** et chaque méta-patron d'extension de **DÉMARCHE** sur chacun de ces concepts, ce qui lui permet de peupler sa bibliothèque de patrons d'extension (Annexe A et B).

Deuxième étape : L'ingénieur de méthodes construit la carte d'extension aux applications temporelles en appliquant les règles de passages entre la bibliothèque de patrons et la carte d'extension (Chapitre V).

Troisième étape : La carte d'extension aux applications temporelles est ensuite appliquée à la méthode O*. L'ingénieur de méthode n'a donc plus qu'à naviguer dans la carte pour étendre O* avec les concepts qui l'intéresse, tant au niveau du **PRODUIT** que de la **DÉMARCHE**. Chaque section de la carte représentant un patron d'extension spécifique (Annexe B) applicable à une méthode orientée objet.

La section 2 propose tout d'abord une analyse succincte des aspects temporels présents dans les applications temporelles. La bibliothèque des méta-patrons est décrite dans l'Annexe A, celle des patrons d'extension dans l'annexe B. La section 3 présente ensuite la carte d'extension associée à l'extension de méthodes aux applications temporelles. Enfin, la section 4 propose la bibliothèque des patrons d'extension instanciés pour la méthode O*.

19 Concepts temporels

Lors de la conception d'une application base de données, l'ingénieur d'application définit quelles sont les classes d'objets pertinentes pour l'application et décrit comment ces objets sont susceptibles d'évoluer. Plusieurs points doivent être étudiés pour décrire les aspects temporels d'une application bases de données.

Quelles sont les unités temporelles qui rythment la vie de l'organisation ?

L'axe du temps est un segment qui peut être divisé en un nombre fini de plus petits segments appelés *granules*.

La plus petite quantité de temps représentable est le *chronon*. La taille d'un instant est égale à la taille du chronon. Une estampille informe qu'un fait s'est passé durant un chronon ou un granule particulier.

L'unité de base internationale est la seconde. Les *calendriers* ont été introduits pour rythmer et mesurer les faits de la vie quotidienne. Un calendrier est une abstraction de l'axe du temps qui est défini par son nom, son origine, une collection de granules ainsi qu'un système de conversion entre les différentes unités.

Beaucoup de méthodes utilisent les mesures du calendrier *Grégorien* pour rythmer la vie d'une organisation étant donné que c'est celui qui est le plus répandu dans le monde. Cependant, il n'est que très rarement défini explicitement. La méthode O* possède les concepts de calendrier et de stimulus temporels qui lui sont rattachés mais ne le définit pas sémantiquement, au contraire des méthodes Dades [Olivé82] et CIAM [Gustafsson82] qui définissent des granularités différentes correspondant à la seconde, à la minute, à l'heure, à la date, au mois et à l'année sans toutefois définir explicitement le calendrier lui-même. Nous avons donc défini des patrons d'extension permettant d'insérer dans les méthodes orientées objets le concept explicite de référentiel temporel.

Quelle est la sémantique et le domaine temporel le plus approprié à chaque attribut (renseignements ayant une sémantique de temps) ?

Trois dimensions temporelles sont présentes dans les bases de données : le temps de validité, le temps de transaction et le temps utilisateur. Le *temps de validité* est le temps où l'information mémorisée dans la base est valide dans le monde réel. Le *temps de transaction* représente l'instant où le fait survenu est enregistré dans la base. Le *temps utilisateur* est nécessaire pour toutes les informations temporelles dépendantes de l'application; il s'agit d'une information ayant une sémantique de temps gérée par l'application.

Il y a trois types temporels possibles : le type instant, le type durée et le type intervalle. Un *instant* représente un point sur l'axe du temps. La taille d'un instant est égale à la taille du chronon. Une *durée* permet de manipuler des quantités de temps qui ne sont pas localisées sur l'axe du temps (*3 mois* ou *15 secondes…)*. Un *intervalle* de temps est une quantité de temps qui est bien située sur l'axe du temps et qui est décrite par un couple d'instants : les bornes inférieure et supérieure.

Il existe trois types de temps : le temps absolu, le temps relatif et le temps périodique. Le *temps absolu* représente un instant fixe ou un intervalle fixe d'un calendrier. Le *temps relatif* représente une référence temporelle liée à une autre référence temporelle. Le *temps périodique* est défini comme une suite infinie d'instants ou d'intervalles (séries temporelles) où la distance temporelle entre deux éléments est exprimée par une durée.

Trois types temporels sont utilisés de façon générale dans les méthodes orientées objets : le type *Date*, le type *Time* et le type *Timezone*. Le modèle Dades [Olivé82] définit de plus les types temporels *Point* et *Interval*, ainsi que des opérations de conversion entre ces deux types. Les estampilles temporelles, excepté le *temps utilisateur*, sont peu présentes dans les méthodes existantes. Il utilise également trois types d'estampilles temporelles : le temps de validité (*assertion time*), le temps intrinsèque (*intrinsic time*) et le temps extrinsèque (*extrinsic time*). Nous avons donc défini des patrons d'extension permettant d'insérer dans les méthodes orientées objets le concept de domaine temporel spécialisable avec ces différents types.

Quelles sont les données dont l'évolution est pertinente à garder et à gérer dans la base de données ?

Associer un temps à un objet signifie y mettre une *estampille*. Elle permet de garder une image de certains objets de la base dans l'état où ils se trouvaient à un moment donné. L'estampillage est utile si

l'on veut associer le temps de modification au dernier état d'un objet. Gérer tous les états d'un objet avec leur estampille temporelle signifie historiser l'objet. L'historisation est utile lorsque tous les états d'un objet doivent être gardés.

Les méthodes classiques ne gèrent pas la gestion de l'évolution des données dans une application. Le modèle Dades [Olivé82] estampille chaque relation pour connaître son temps de validité. Dans DCM [Olivé89], toute information est associée à un point du temps. Nous avons donc défini des patrons d'extension permettant d'insérer dans les méthodes orientées objets le concept d'historiques d'objets et celui d'estampillage d'objets.

La sémantique de temps des stimulus externes peut également être utile dans certaines applications et est en conséquence pris en compte dans un certain nombre de méthodes. Dans beaucoup de méthodes, ces stimulus sont perçus comme des messages permettant d'invoquer des opérations. Nous avons donc défini des patrons d'extension permettant d'insérer dans les méthodes orientées objets le concept d'estampillage des stimulus externes.

Quelles sont les contraintes à définir pour pouvoir assurer la cohérence des états passés, présent ou futurs des données temporelles ?

Si l'ingénieur d'application a jugé nécessaire l'utilisation de données temporelles, il doit également prendre en compte la définition des contraintes gérant l'intégrité de la base de données par rapport à ces données.

Peu de méthodes gèrent l'historique de l'évolution des données, en conséquence, peu de méthodes également gèrent les contraintes temporelles inhérentes à ces problèmes. Le modèle Dades [Olivé82] ordonne les occurrences de ses relations selon le temps de validité qu'il leur a assigné. Les méthodes OSA [Embley92] et ERAE définissent toutes deux une contrainte de temps réel. Dans ERAE, ces contraintes sont réglées avec des opérateurs temporels modaux. La méthode ALBERT [Dubois94] utilise des contraintes temporelles pour séquencer les actions définies dans l'application (utilisation d'opérateurs modaux). Nous avons donc défini des patrons d'extension permettant d'insérer dans les méthodes orientées objets le concept de contraintes temporelles.

20 Carte d'extension aux applications temporelles

Les problèmes définis précédemment peuvent être gérés dans l'extension temporelle d'une méthode en y introduisant, par exemple, des concepts permettant de mesurer et de représenter le temps ou la gestion d'historiques. Ces concepts permettent de prendre en compte les aspects temporels d'une application à un niveau conceptuel.

L'importance du temps dans le domaine des bases de données est évidente depuis 1975. Les applications nécessitent de gérer des données passées, présentes et futures. Les bases de données conventionnelles sont construites pour stocker et retrouver des données courantes. Lors des opérations de modification, les données sont écrasées et remplacées par leur nouvelles valeurs. Cela signifie que

les données passées ne sont plus disponibles dans la base de données et que seules les données courantes sont stockées. Cela justifie l'addition de mécanismes dédiés à modéliser, retrouver, stocker et modifier des données dépendantes du temps. Ceci a induit l'introduction de SQL/T (extension temporelle de SQL) dans SQL3. Cette nouvelle génération de SGBD temporel intègre le temps dans le modèle de données pour faciliter la gérance des aspects temporels au niveau applicatif. De tels SGBD ont simplifié le stockage et la recherche des données temporelles. Cependant, ils n'ont pas résolu les problèmes de conception comme la définition du schéma de données, la spécification des activités de l'application, la vérification de la consistance de la base de données et ne fournissent pas de règle de conduites pour administrer la base de données. L'introduction de caractéristiques temporelles dans les modèles OO est toujours en cours de recherche.

Nous avons identifié, à travers le projet TOOBIS[9], un ensemble de besoins temporels pour les applications de base de données temporelles. Pour chacun de ces problèmes temporels, nous avons défini une solution générique que nous avons encastrée dans un patron. Chacun de ces patrons guide l'ingénieur dans le processus d'adaptation de son modèle OO pour gérer les caractéristiques temporelles associées au patron.

La façon de naviguer entre les exécutions de patrons est décrite par la carte d'extension illustrée à la Figure 161 suivante.

[9] TOOBIS (Projet ESPRIT No. 20671) est un acronyme pour « Temporal Object and Information Systems ». Ce projet a permis de définir un SGBD Objet basé sur O_2 avec une méthode de conception spécifique.

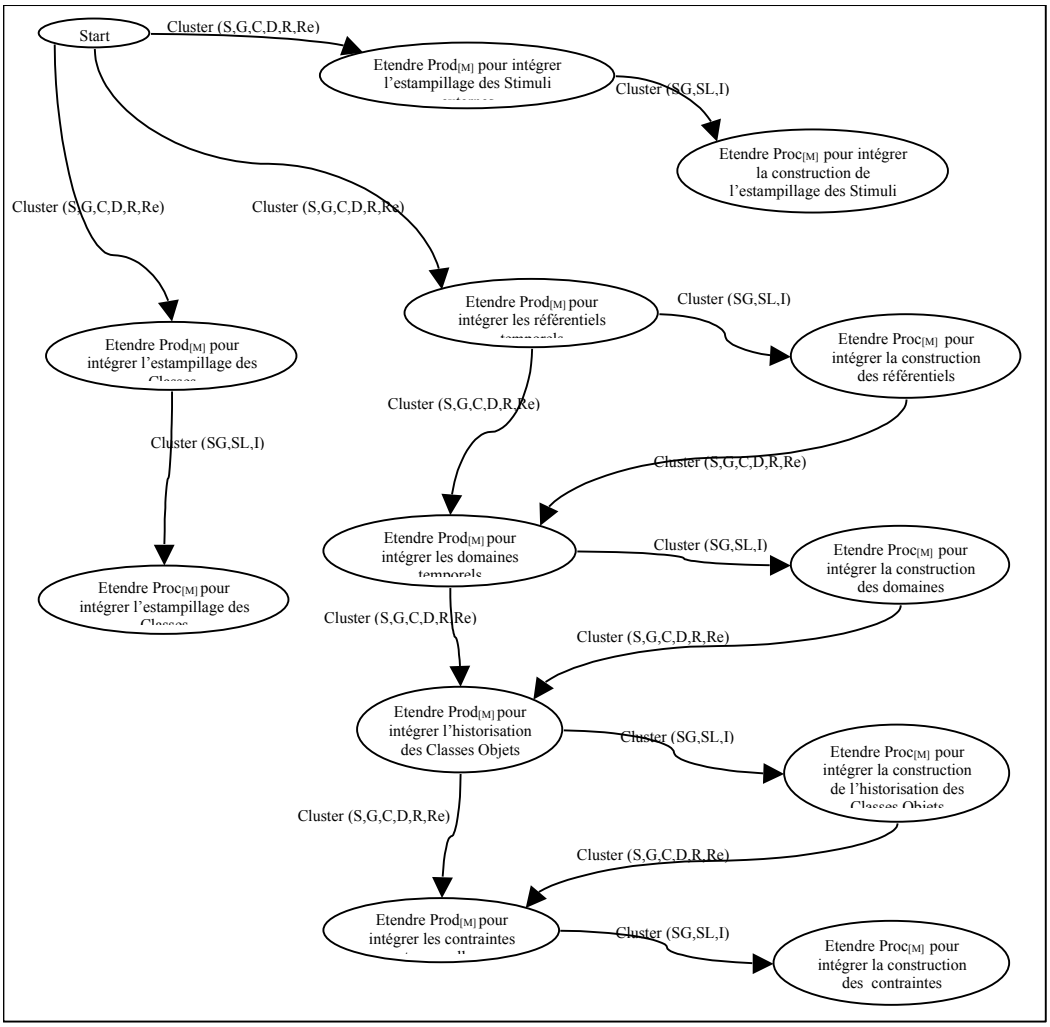


Figure 161: Carte d'extension d'une méthode aux applications temporelles

La bibliothèque des patrons d'extension associés à cette carte est donnée à l'annexe B.

21 Exemple d'application de cette carte d'extension sur la méthode O*

Nous avons choisi comme méthode d'origine la méthode O* et avons décidé d'effectuer une extension complète de cette méthode, tant au niveau **PRODUIT** qu'au niveau **DÉMARCHE** en y intégrant le plus de fonctionnalités possibles. Pour cela, il nous faut donc exécuter chacune des intentions de la carte d'extension. Le chemin utilisé, les problèmes pris en compte, le choix de telle ou telle stratégie pour atteindre ces intentions sont décrits dans les sections suivantes.

21.1 Insertion du concept de référentiel temporel dans O*

21.1.1 Problème pris en compte

Le concept que l'on utilise habituellement pour mesurer le temps est celui de calendrier. Un calendrier est un système métrique que l'on applique sur l'axe du temps. Traditionnellement, les bases de données se basent sur un seul calendrier, le calendrier *Grégorien* (représenté généralement par l'horloge interne). Les types temporels *Date* ou *Datetime* permettent de représenter des estampilles temporelles avec des granules grégoriens différents (jour ou seconde). Mais, si l'organisation possède des rythmes temporels spécifiques, l'application doit explicitement les gérer. Prenons l'exemple d'une petite entreprise : si le comptable rythme ses activités avec le calendrier *Comptable* alors que le chef de projet utilise le calendrier des *Jours Ouvrés*, l'application devra gérer ces deux calendriers en même temps (avec les opérateurs de translation et de conversion qui leur sont associés).

21.1.2 Choix du patron d'extension

L'extension d'une méthode par l'insertion d'un concept de référentiel temporel est représentée dans la carte d'extension comme l'intention cible de six sections différentes ayant chacune la même intention source *(Démarrer)*. Cela signifie qu'il existe six stratégies différentes pour accéder à cette intention, ainsi que le montre la Figure 162.

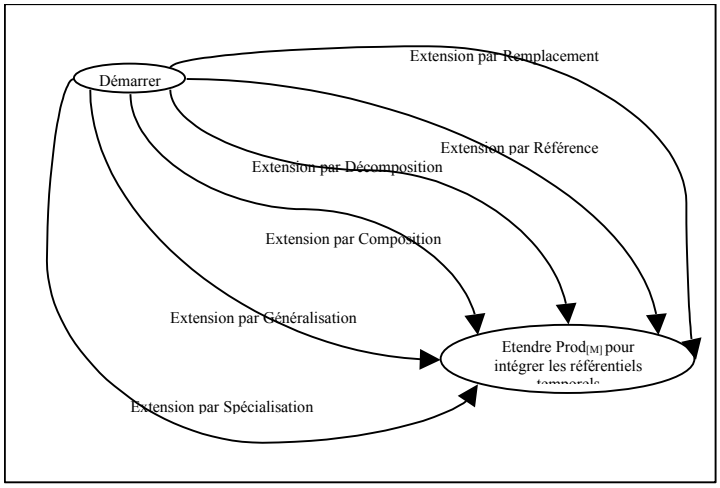


Figure 162: Sections de la carte d'extension d'une méthode aux applications temporelles ayant comme intention cible l'insertion du concept de référentiel temporel

Le choix de la stratégie allant de l'intention *Démarrer* à l'intention *Etendre $Prod_{[M]}$ pour intégrer les référentiels temporels*, c.-à-d. le patron à appliquer, est dépendant de la situation de la méthode d'origine.

Le modèle de la méthode O* possède déjà un concept de référentiel temporel implicite, la *Classe Horloge*. Cependant, au niveau objet, ce concept ne représente en fait qu'une seule instance de référentiel temporel, le calendrier *Grégorien*.

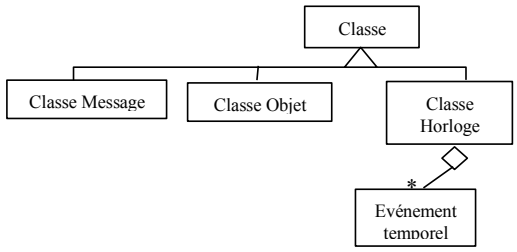
Description du **PRODUIT** avant extension	Représentation du **PRODUIT** avant extension
Classe Horloge = **Composition** ({Evénement temporel}) Classe = **Spécialisation** (Classe Objet, Classe Message, Classe Horloge)	

Figure 163 : Partie du PRODUIT de O* avant l'application du patron

On peut voir sur la Figure 163 que le concept de *Classe Horloge* est une spécialisation de celui de *Classe* possédant également deux autres spécialisations possibles : la *Classe Message* et la *Classe Objet*. De plus, cette *Classe Horloge* est une composition d'instances du concept d'*Evénement Temporel*.

Comme la méthode O* possède déjà un concept de référentiel temporel qui est défini de façon incomplète pour les besoins de l'ingénieur de méthodes, mais que les liens de ce concept avec les autres éléments du modèle sont corrects (lien de spécialisation avec *Classe* et lien de composition avec *Evénement Temporel*), l'ingénieur va appliquer sur ce modèle la stratégie d'extension par REMPLACEMENT.

L'ingénieur de méthodes exécute donc le patron d'extension décrit dans l'annexe B dont l'intention est d'*« Etendre, par REMPLACEMENT, le concept de Classe Horloge pour intégrer le concept de Classe Calendrier ».*

21.1.3 Instanciation du patron d'extension à la méthode O*

L'interface du patron d'extension instancié pour la méthode O* est la suivante.

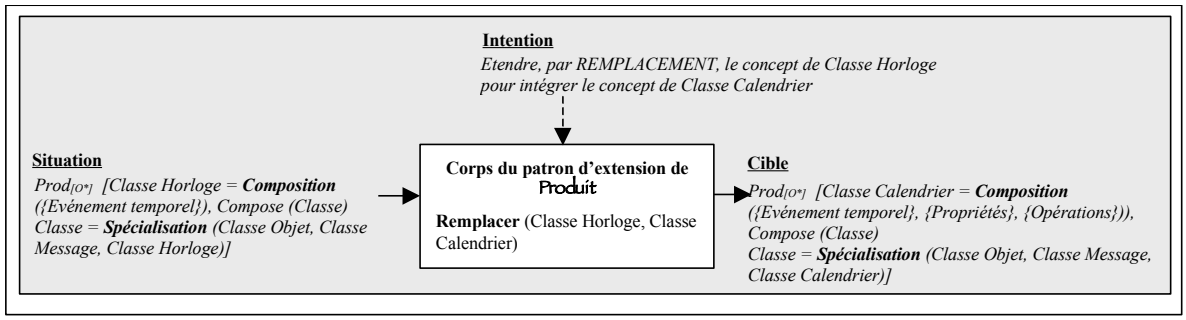


Figure 164: Application du patron d'extension de la méthode O* permettant de remplacer le concept de référentiel temporel

Le résultat de l'application de ce patron au niveau du **PRODUIT** de la méthode est une description de *Classe Calendrier*. Le corps du patron guide la génération de cette description en utilisant un modèle de calendrier. La description de la *Classe Calendrier* est une composition de deux parties : *Propriété* et *Opération*. Les *Propriété(s)* comprennent l'origine du calendrier (instant bornant à gauche l'axe du temps) et une liste ordonnée de granules, de différents niveaux d'abstraction, construites à partir d'un

granule de base, contenant, pour chacun d'eux, la conversion ascendante et la conversion descendante. Lorsque la conversion est régulière, on précise le coefficient à appliquer mais lorsque elle est irrégulière, l'irrégularité doit être prise en compte dans les opérations de translation et de conversion. Les *Opération(s)* correspondent à l'ensemble des opérations de conversion : une opération de translation d'un instant en une période (durée), une opération de conversion entre granules d'un même calendrier et deux opérations de conversion, l'une vers le calendrier *Grégorien* et l'autre partant du calendrier *Grégorien*. Ces deux dernières opérations permettent d'avoir des conversions inter calendriers.

Ce patron fournit alors une *Classe Calendrier* qui peut être instanciée plusieurs fois pour obtenir plusieurs calendriers. De plus, le patron fournit également un calendrier par défaut qui est le calendrier *Grégorien* (instance de la *Classe Calendrier*, définie à la Figure 165). Si le modèle orienté objet ne prend pas cette nécessité en compte, le patron l'adaptera à cet usage. Par exemple, une entreprise peut utiliser le calendrier *Grégorien* dans tous ses départements et le calendrier des *Jours Ouvrés* pour le département du personnel. En conséquence, il est nécessaire de définir deux calendriers dans l'application (et bien entendu les opérations de conversions entre ces deux calendriers).

Classe Calendrier Grégorien
origine "01/01/0001:00:00:00" **unité de base** = chronon
granules
seconde : {1 unité de base, unitéde base =1 seconde}
minute : {irrégulier}
heure : {60 minutes, minute=1/60 heure}
jour : {24 heures, heure =1/24 jour}
mois : {irrégulier}
année : {12 mois, mois =1/12 année}
opérations
instant<Grégorien, *> opérateur+ (instant<Grégorien, *>, intervalle<Grégorien, *>)
instant<Grégorien, *> opéraeur+ (intervalle<Grégorien, *>, instant<Grégorien, *>)
interval<Grégorien, *> opérateur- (instant<Grégorien, *>, instant<Grégorien, *>)
FinClasseCalendrier

Figure 165 : Définition du calendrier Grégorien.

La minute et le mois sont des granules irréguliers car leur quantité de temps n'est pas fixe. En effet, la minute représente habituellement 60 secondes, excepté lorsqu'on lui ajoute une seconde intermédiaire. De la même manière, le nombre de jours contenus dans un mois est variable, il va de 28 à 31 jours selon le mois considéré.

Le corps du patron est donc composé d'un opérateur permettant le remplacement de l'ancienne *Classe Horloge*, ainsi que le montrent la Figure 166.

Corps du Patron	Description du **PRODUIT** après extension
Remplacer (Classe Horloge, Classe Calendrier)	Classe = **Spécialisation** (Classe Objet, Classe Message, Classe Calendrier) Classe Calendrier = **Composition** ({Evénement temporel}, {Propriétés}, {Opérations}), **Compose** (Classe) Propriétés = **Composition** (Origine, {Granule}), **Compose** (Classe Calendrier)

Figure 166: Partie du PRODUIT de O* après l'application du patron

On peut voir sur la Figure 46 que la partie **PRODUIT** de la méthode O* a été étendue pour intégrer une nouvelle définition du concept de *Classe Calendrier* pour permettre à l'ingénieur de méthodes de définir plusieurs référentiels temporels.

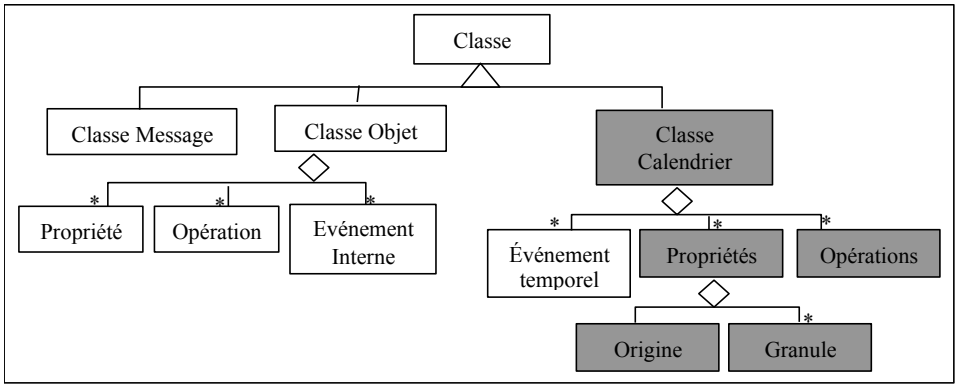


Figure 167: Partie du méta-modèle de la méthode O* étendue

On peut constater sur cette figure et sur la description du **PRODUIT** après extension de la Figure 166 que l'insertion du concept de *Classe Calendrier* s'accompagne de l'insertion de toute sa spécification, c'est à dire de sa composition avec les concepts de *Propriété* et *Opération*. De la même manière, la spécification du concept de *Propriété* en composition avec les concepts d'*Origine* et de *Granule* est également prise en compte.

21.2 Insertion de la construction du concept de référentiel temporel dans O*

21.2.1 Problème pris en compte

L'ingénieur de méthodes a intégré le concept de référentiel temporel dans la partie **PRODUIT** de la méthode d'origine et il souhaite maintenant étendre la partie **DÉMARCHE** de celle-ci.

21.2.2 Choix du patron d'extension

L'insertion de la construction du concept de référentiel temporel dans la partie **DÉMARCHE** d'une méthode est représentée dans la carte d'extension comme l'intention cible de trois sections ayant la même intention source *(Démarrer)*. Ces trois sections signifient qu'il y a trois stratégies différentes permettant d'atteindre cette intention, ainsi que l'indique la Figure 168.

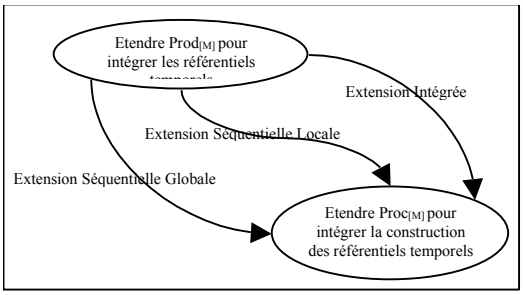


Figure 168: Sections de la carte d'extension d'une méthode aux applications temporelles ayant comme intention cible l'insertion de la construction du concept de référentiel temporel

Le choix de la stratégie permettant de naviguer entre l'intention de départ *Démarrer* et l'intention qui nous intéresse *Etendre Proc[M] pour y intégrer la construction des référentiels temporels*, c.-à-d. le patron à appliquer, est dépendant de la situation de la méthode d'origine, ici O*.

Le modèle de la méthode O* possède maintenant le concept de *Classe Calendrier* qui a été inséré par REMPLACEMENT. Cependant, la partie **DÉMARCHE** de O* n'a pas encore pris cette modification en compte. La Figure 169 illustre la description de la **DÉMARCHE** de O* avant toute extension, à la fois par une description textuelle et par une représentation graphique.

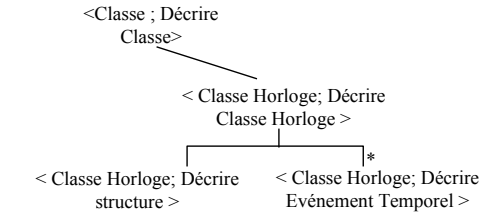
Description de la **DÉMARCHE** avant extension	Représentation de la **DÉMARCHE** avant extension
Parent-de($Proc_{[Classe\ Horloge]}$) = <Classe ; Décrire Classe> $Proc_{[Classe\ Horloge]}$ = <Classe Horloge ; Décrire Classe Horloge> ; $Proc_{[W]}$ = <Evénement Temporel ; Décrire Evénement Temporel >	

Figure 169 : Partie de la DÉMARCHE de O* avant l'application du patron

La **DÉMARCHE** de construction du concept de *Classe Horloge* dans l'arbre de processus de la méthode O* est représentée par un plan permettant d'exécuter deux étapes : la description des *Evénements Temporels* liés au calendrier et la description des liens que ce concept a avec le reste du modèle.

La seule stratégie d'extension de la **DÉMARCHE** possible après une exécution de la stratégie d'extension par REMPLACEMENT est la stratégie d'extension INTEGREE. On applique donc le patron d'extension correspondant à la stratégie d'extension par REMPLACEMENT INTEGRE.

L'ingénieur de méthodes exécute donc le patron d'extension décrit dans l'annexe B et dont l'intention est *« Etendre, par REMPLACEMENT INTEGRE, la construction du concept de Classe pour intégrer la construction du concept de Classe Calendrier ».*

21.2.3 Instanciation du patron d'extension à la méthode O*

L'interface du patron d'extension instancié pour la méthode O* est la suivante.

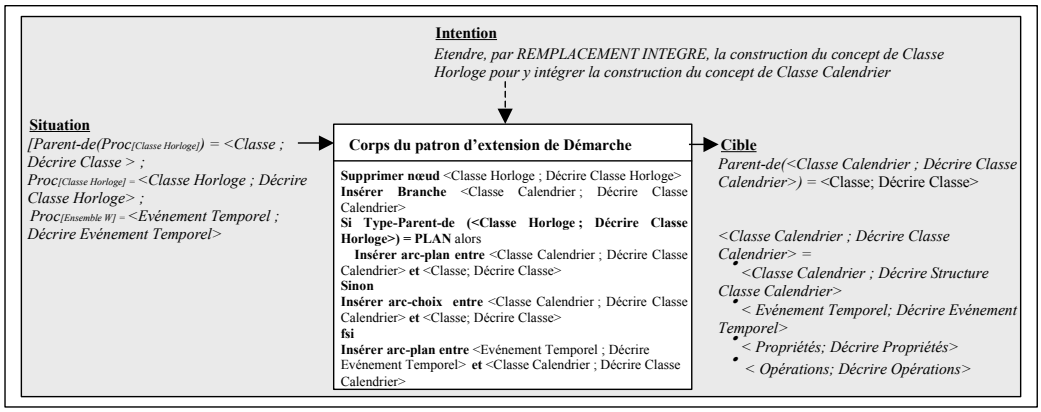


Figure 170: Application du patron d'extension de la méthode O* permettant d'intégrer la construction du concept de Classe Calendrier dans la DÉMARCHE.

L'application de ce patron au niveau de la **DÉMARCHE** de la méthode O* permet de remplacer la démarche de description de la *Classe Horloge* avec la démarche de description de la *Classe Calendrier*. Cependant, la démarche de description des liens que l'ancien concept avait avec le reste du modèle reste inchangée et est conservée pour la *Classe Calendrier*.

Le corps de ce patron, composé d'un ensemble d'opérateurs permettant le remplacement de cette démarche est décrit à la Figure 171, ainsi que la description textuelle de cette partie de la Démarche de O* après exécution de ces opérateurs.

Corps du Patron	Description de la **DÉMARCHE** après extension
Supprimer nœud <Classe Horloge ; Décrire Classe Horloge> **Insérer Branche** <Classe Calendrier ; Décrire Classe Calendrier> **Si Type-Parent-de** (<Classe Horloge ; Décrire Classe Horloge>) = PLAN alors **Insérer arc-plan entre** <Classe Calendrier ; Décrire Classe Calendrier> **et** <Classe; Décrire Classe> **Sinon** **Insérer arc-choix entre** <Classe Calendrier ; Décrire Classe Calendrier> **et** <Classe; Décrire Classe> **fsi** **Insérer arc-plan entre** <Evénement Temporel ; Décrire Evénement Temporel> **et** <Classe Calendrier ; Décrire Classe Calendrier>	Parent-de(<Classe Calendrier ; Décrire Classe Calendrier>) = <Classe ; Décrire Classe> <Classe Calendrier ; Décrire Classe Calendrier> = • < Evénement Externe; Décrire Evénement Externe> • < Propriétés; Décrire Propriétés> • < Opérations; Décrire Opérations>

Figure 171: Partie de la DÉMARCHE de O* après l'application du patron

On peut donc constater que la partie **DÉMARCHE** de la méthode O* a été étendue de manière à prendre en compte la construction du nouveau concept de *Classe Calendrier* à la place de la construction de la *Classe Horloge*. La Figure 172 donne la représentation graphique de cette partie de la **DÉMARCHE**.

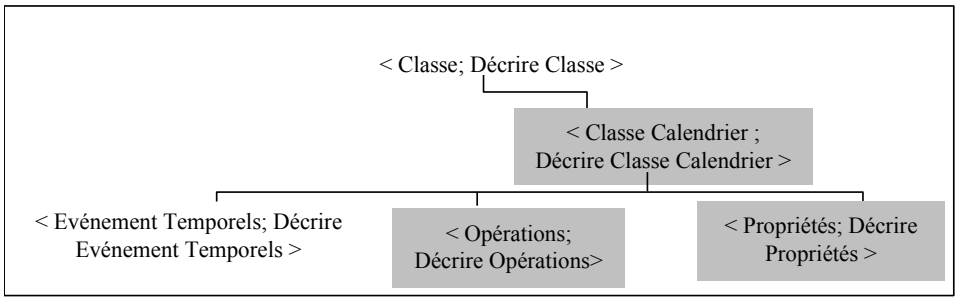


Figure 172: Partie de l'arbre de processus de la méthode O* étendue

21.3 Insertion du concept de domaine temporel dans O*

21.3.1 Problème pris en compte

Dans les modèles orientés objets classiques, la seule façon de doter un attribut d'un domaine temporel est de sélectionner l'un des types suivants : *Date*, *Time* ou *Timezone*. Cependant, cet ensemble de types n'est pas toujours suffisant par rapport aux applications. Une période de temps (souvent modélisée avec deux attributs Date *Début* et *Fin*) ou une durée (souvent un type numérique) sont des abstractions qu'il est parfois utile d'utiliser avec leurs opérateurs associés. Par exemple, si l'on veut ajouter 20 jours à un intervalle ou si l'on veut ajouter 10 mois à une période etc..

De la même manière, il est parfois nécessaire de gérer non seulement les temps absolus mais également les temps relatifs. Par exemple, la date de début d'une tâche peut être exprimé relativement à la fin d'une tâche précédente qui n'est pas encore terminée (par exemple 20 jours après).

21.3.2 Choix du patron d'extension

L'insertion du concept permettant la représentation des domaines temporels est illustrée sur la carte comme l'intention cible de douze sections différentes. Six de ces sections ont comme intention source *Démarrer*, les six autres *Etendre Proc[M] pour intégrer la construction des référentiels temporels*. Cela signifie qu'il y a six stratégies possibles entre cette intention et notre intention cible ainsi que six stratégies possibles entre *Démarrer* et notre intention cible, ainsi que le visualise la figure suivante.

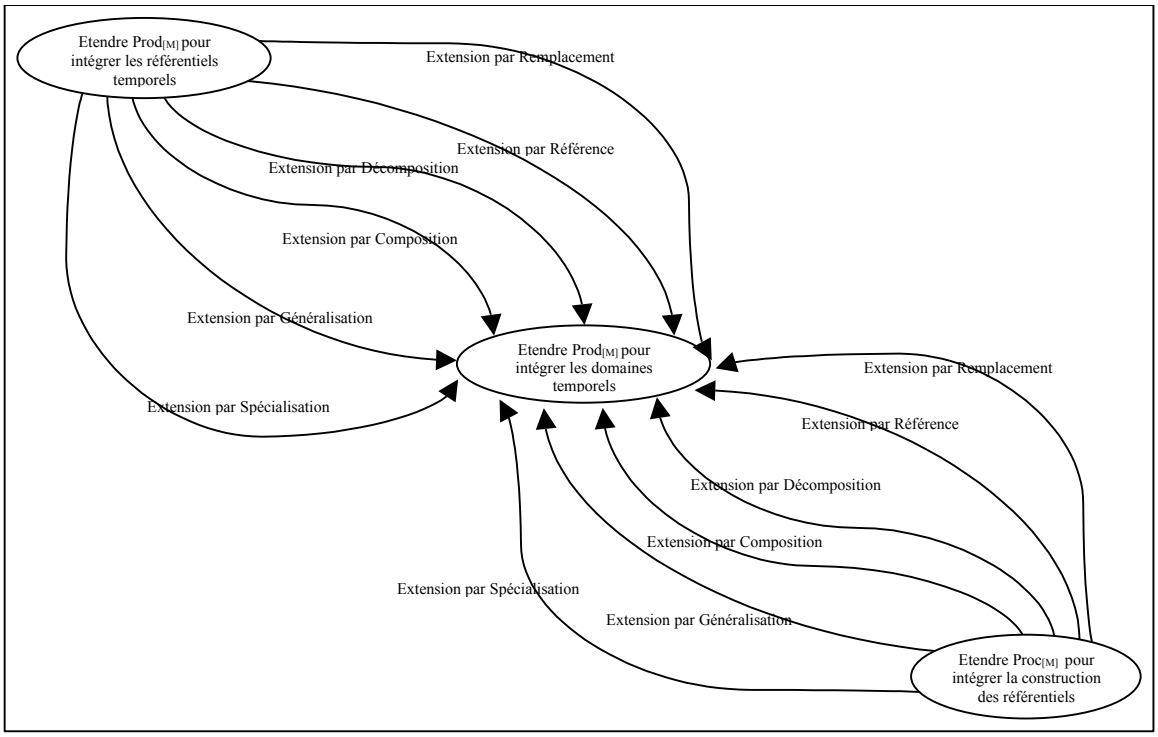


Figure 173: Sections de la carte d'extension d'une méthode aux applications temporelles ayant comme intention cible l'insertion des domaines temporels

Le choix de la stratégie allant de l'une de ces intentions à notre intention cible (le patron à appliquer pour étendre la méthode) dépend de la situation en cours de O*.

Le modèle possède uniquement le concept de *Date* pour décrire une donnée représentant du temps. La Figure 174 décrit la partie du **PRODUIT** de la méthode O* correspondant au concept de *Domaine*, tant par une représentation textuelle que graphique. Cette méthode contient également le concept de *Granule*, concept ayant été intégré dans le **PRODUIT** lors de l'extension de la méthode aux référentiels temporels.

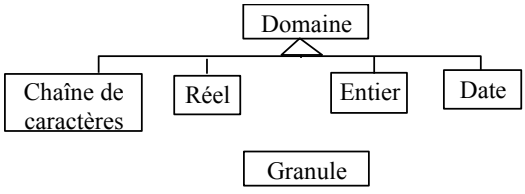
Description du **PRODUIT** avant extension	Représentation du **PRODUIT** avant extension
Domaine = **Spécialisation** (Chaîne de caractères, Réel, Entier, Date) Granule	

Figure 174 : Partie du PRODUIT de O* avant l'application du patron

Le concept de *Domaine* présent dans la méthode O* est une généralisation des concepts de domaines simples usuels tels que les entiers, les réels, les chaînes de caractères et les Dates.

La stratégie à appliquer sur un produit de ce type est celle de la GENERALISATION pour que l'on puisse insérer un nouveau concept de *Domaine* qui possédera deux généralisations : *Domaine non*

Temporel (l'ancien concept de *Domaine*) et *Domaine Temporel*. Il faudra également supprimer l'ancien concept *Date* pour ne pas provoquer d'incohérence dans la méthode.

L'ingénieur de méthodes exécute donc le patron d'extension décrit dans l'annexe B et dont l'intention est d'*« Etendre, par GENERALISATION, le concept de Domaine pour intégrer le concept des domaines temporels».*

21.3.3 Instanciation du patron d'extension à la méthode O*

L'interface du patron d'extension instancié pour la méthode O* est la suivante.

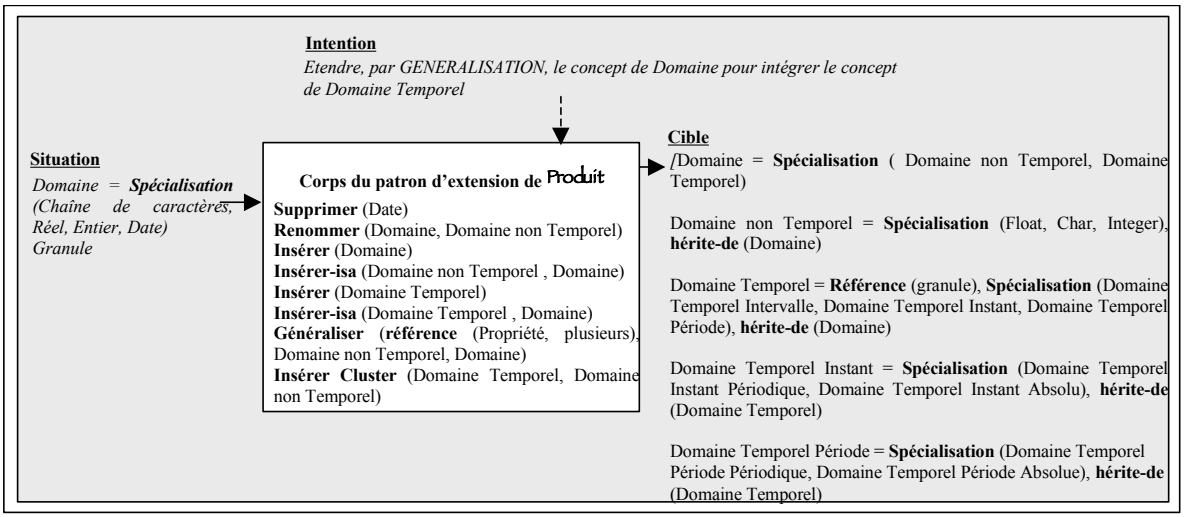


Figure 175: Application du patron d'extension de la méthode O* permettant de remplacer le concept des domaines temporels

Le modèle obtenu par l'application du patron permet de définir trois différents types de *Domaine Temporel* : les instants, les périodes et les intervalles. Un point isolé sur l'axe du temps est appelé un *Instant (12/12/1996)*. Le temps entre deux instants est appelé un *Intervalle* de temps *([12/12/1996, 15/12/1996]*). Un intervalle de temps est précisément décrit par un couple d'instants : les bornes inférieure et supérieure. Un instant peut être considéré comme un intervalle de durée nulle (la borne inférieure est égale à la borne supérieure). Les bornes d'un intervalle peuvent être incluses ou pas. Une *Période* de temps est définie comme une durée *(3 mois)*, c'est à dire une quantité de temps qui n'est pas localisable sur l'axe du temps. Par opposition, un intervalle de temps est une quantité de temps qui est bien située sur l'axe du temps. Ces nouveaux domaines temporels remplacent respectivement les types *Date*, *Time* et *Datetime* que l'on trouve dans les bases de données classiques.

Le modèle obtenu permet aussi de raisonner en temps absolu ou en temps relatif. Ceci est réalisé par une spécialisation des domaines temporels *Instant* et *Intervalle*.

- *Instant-A* et *Intervalle-A* permettent de représenter les instants ou les intervalles de temps absolu,
- *Instant-R* et *Intervalle-R* sont les domaines représentant les instants ou les intervalles de temps relatif,

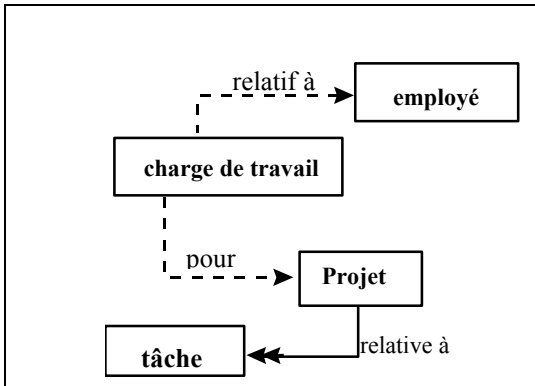
- *Instant* et *Intervalle* sont conservés pour représenter des instants ou des intervalles dont le type de temps n'est pas fixé à l'avance.

Les domaines temporels sont nécessaires pour définir le domaine de valeurs des attributs ayant une sémantique temporelle. Par exemple, l'attribut *datenaissance* de la classe *Employé* se décrit comme un instant exprimé au niveau jour du calendrier *Grégorien* comme l'illustre la Figure 176.

Classe Employé
propriétés
datenaissance : **INSTANT** <Grégorien, jour>

Figure 176 : Exemple d'attribut temporel.

Un autre exemple est donné à la Figure 177 suivante.



Classe : Employé
Propriétés
Nom : **Chaîne**
DateNaissance : **INSTANT** < *Grégorien, jour* >

Classe : Charge de travail
Propriétés
Charge : **PERIODE** <*Grégorien, heure* >
Sem : **INSTANT** <*MaSemaine , semaine* >
Relatif à : **ref** (Employé)
Pour : **ref** (Projet)

Classe : Projet
Propriétés
Nom : **Chaîne**
Dates : **INTERVALLE** <*Grégorien, jour*>
Début du projet : **INSTANT-A** <*Grégorien, jour*>
Fin du projet : **INSTANT-R** <*Grégorien, jour*>
Tâches : **comp** (ensemble (tâche))

Classe : Tâche
Propriétés
Nom de la tâche : **Chaîne**
Dates : **INTERVALLE-R** <*Grégorien, jour*>
Objectif : **Chaîne**
Descriptif : **Chaîne**

Figure 177 : Exemples de classes utilisant plusieurs calendriers et plusieurs domaines temporels.

Nous pouvons voir dans cet exemple que nous pouvons définir comme temps absolu le temps des attributs temporels D*ateNaissance*, *Sem*. Par contre, l'intervalle de temps du projet (attribut *Dates*) reste de type *Intervalle* (type de temps non déterminé) parce que certains projets pourraient l'exprimer de manière absolue (comme par exemple durant l'intervalle de temps [97/10/02 ; 98/06/02)) et d'autres de manière relative (comme par exemple entre [date_signature ; date_signature+18mois)).

Par exemple, la date de fin du projet est égale à "J0+30mois"; J0 étant un repère temporel. De même, la tâche1 de ce projet se déroule durant l'intervalle en temps relatif suivant : ["début du projet+ 7 jours", "début du projet+2mois7jours"). Le temps relatif s'apparente à un attribut temporel dérivable ou calculé. La formule de calcul est limitée à l'addition du repère temporel à une durée (<repère temporel> + <durée>). La particularité de l'attribut temporel relatif, est qu'il n'a pas de valeur tant que la valeur du repère temporel n'est pas connue. Le repère temporel est un attribut temporel qui s'exprime lui-même de manière relative ou absolue.

L'attribut temporel *début du projet* est un instant absolu exprimé en jours alors que les attributs temporels relatifs *fin du projet* et *dates* sont respectivement de type *Instant* et *Intervalle*. Par exemple, le projet *DBT* a un instant de fin déterminé par la formule : "début du projet + 6 mois". C'est un instant relatif à un autre attribut temporel *début du projet* qui est un instant absolu. Le projet *DBT* est décomposé en 2 tâches. Les dates de la première tâche sont déterminées par l'intervalle relatif suivant : ["début du projet + 7 jours", "début du projet + 2mois7jours"); et la deuxième tâche se déroule durant l'intervalle relatif : ["(tâche1()).dates.fin + 7 jours", "(tâche1()).dates.fin + 3mois7jours"). Le repère temporel utilisé est *la fin de la tâche1*, lui-même relatif. La fonction *tâche1()* correspond à la requête SQL permettant d'obtenir l'objet *tâche1* du projet *DBT* à partir de la *tâche2*.

Le résultat de l'application du patron est donc la définition d'un nouvel ensemble de domaines : les domaines temporels. Ces types sont exprimés avec la granularité d'un calendrier.

Corps du Patron	Description du **PRODUIT** après extension
Supprimer (Date) **Renommer** (Domaine, Domaine non Temporel) **Insérer** (Domaine) **Insérer-isa** (Domaine non Temporel , Domaine) **Insérer** (Domaine Temporel) **Insérer-isa** (Domaine Temporel , Domaine) **Généraliser** (**référence** (Propriété, plusieurs), Domaine non Temporel, Domaine) **Insérer Cluster** (Domaine Temporel, Domaine non Temporel)	Domaine = **Spécialisation** (Domaine non Temporel, Domaine Temporel) Domaine non Temporel = **Spécialisation** (Float, Char, Integer), **hérite-de** (Domaine) Domaine Temporel = **Référence** (granule), **Spécialisation** (Domaine Temporel Intervalle, Domaine Temporel Instant, Domaine Temporel Période), **hérite-de** (Domaine) Domaine Temporel Instant = **Spécialisation** (Domaine Temporel Instant Périodique, Domaine Temporel Instant Absolu), **hérite-de** (Domaine Temporel) Domaine Temporel Période = **Spécialisation** (Domaine Temporel Période Périodique, Domaine Temporel Période Absolue), **hérite-de** (Domaine Temporel)

Figure 178: Partie du PRODUIT de O* après l'application du patron

On peut voir sur la Figure 46 que la partie **PRODUIT** de la méthode O* a été étendue pour intégrer une nouvelle définition du concept de domaine pour permettre à l'ingénieur de méthodes de définir des domaines temporels.

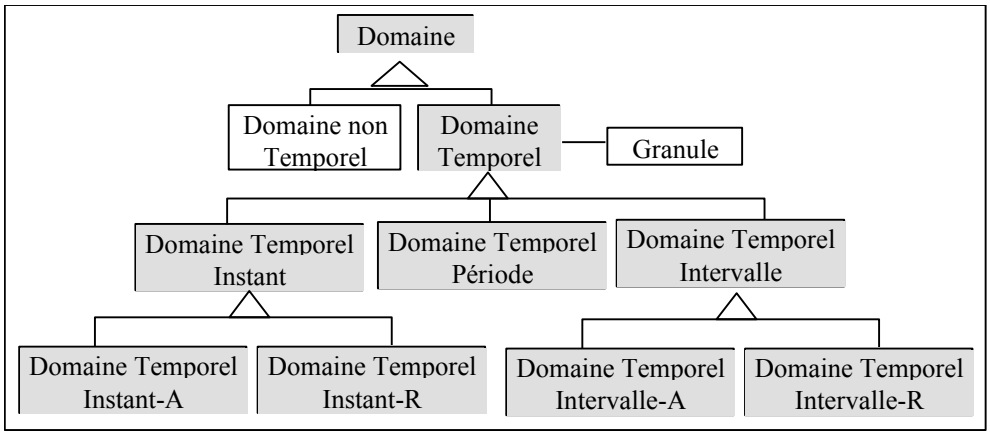


Figure 179: Partie du méta-modèle de la méthode O* étendue

21.4 Insertion de la construction des domaines temporels dans O*

21.4.1 Problème pris en compte

L'ingénieur de méthodes a intégré le concept de domaine temporel dans la partie **PRODUIT** de la méthode d'origine et il souhaite maintenant étendre la partie **DÉMARCHE** de celle-ci pour pouvoir y intégrer la construction de ce concept.

21.4.2 Choix du patron d'extension

L'insertion de la construction des concepts spécifiques aux domaines temporels dans une méthode OO est représentée dans la carte d'extension comme l'intention cible de trois sections spécifiques ayant chacune la même intention source *(Démarrer)*. Ces trois sections différentes représentent chacune une stratégie particulière permettant d'aller de l'intention *Démarrer* à notre intention cible représentant notre objectif.

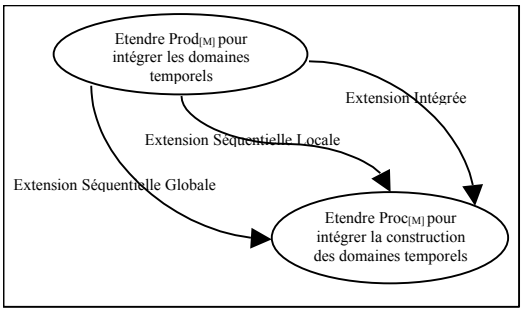


Figure 180: Sections de la carte d'extension d'une méthode aux applications temporelles ayant comme intention cible l'insertion de la construction des domaines temporels

Le choix de la stratégie allant de l'intention source à l'intention cible se définit selon la situation en cours de la méthode d'origine, ici O*. Nous pouvons voir sur la figure suivante que la partie de la **DÉMARCHE** de O* concernant la description des *Domaine(s)* fait partie d'une démarche la mettant en séquence avec l'identification des *Domaine(s)*.

Description de la **DÉMARCHE** avant extension	Représentation de la **DÉMARCHE** avant extension
<Elément ; Décrire Elément> = " < Domaine; Décrire Domaine >	< Elément ; Décrire Elément > < Domaine; Décrire Domaine >

Figure 181 : Partie de la DÉMARCHE de O* avant l'application du patron

Le modèle de la méthode O* possède maintenant le concept de *Domaine Temporel* qui y a été intégré par la stratégie GENERALISATION. Cependant, la partie **DÉMARCHE** de O* n'a pas encore pris cette modification en compte. Différentes stratégies d'extension de la **DÉMARCHE** sont possibles après cette stratégie d'extension du **PRODUIT.** Comme l'ingénieur de méthodes a commencé son exécution de l'extension de la **DÉMARCHE** de O* par la stratégie INTEGREE, il paraît assez logique de continuer ainsi et d'intégrer toute modification de la **DÉMARCHE** de cette manière. De

même, comme nous avons déjà privilégié préalablement la spécialisation par type à la spécialisation par état, nous allons continuer de la même façon. La stratégie à appliquer pour cette extension spécifique sera celle de l'extension par GENERALISATION INTEGREE (par type).

L'ingénieur de méthodes exécute donc le patron d'extension décrit dans l'annexe B et dont l'intention est *« Etendre, par GENERALISATION INTEGREE (par type), la construction du conept de Domaine pour intégrer la construction des domaines temporels ».*

21.4.3 Instanciation du patron d'extension à la méthode O*

L'interface du patron d'extension instancié à O* est la suivante.

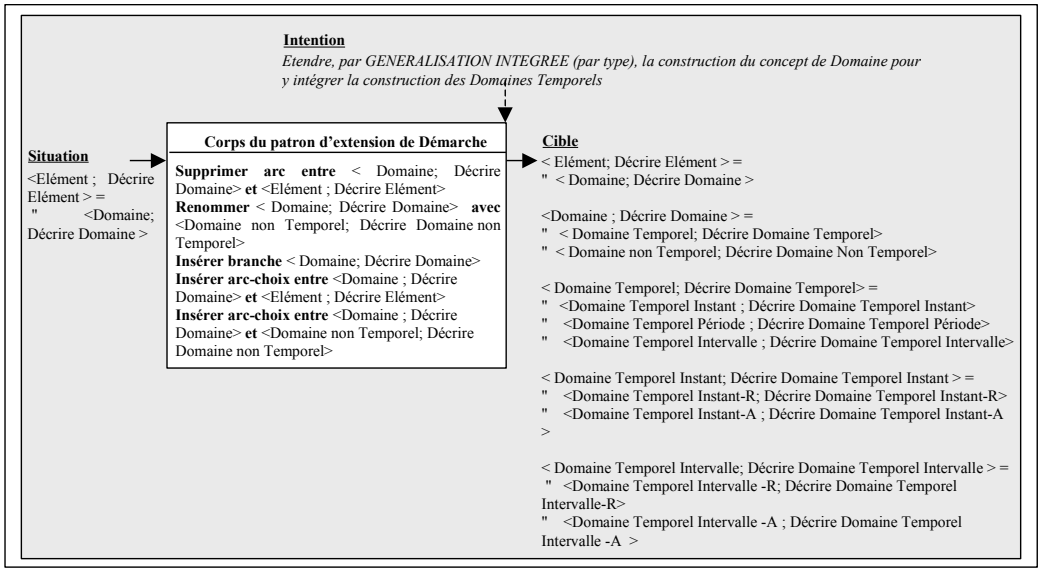


Figure 182: Application du patron d'extension de la méthode O* permettant d'intégrer la construction des domaines temporels.

La **DÉMARCHE** obtenue après extension permet donc de décrire à la fois les domaines temporels et les domaines non temporels.

Le corps du patron est composé d'un ensemble d'opérateurs permettant de modifier la partie **DÉMARCHE** de la méthode O* pour prendre en compte la description de ces domaines temporels.

Corps du Patron	Description de la **DÉMARCHE** après extension
Supprimer arc entre < Domaine; Décrire Domaine> **et** <Elément ; Décrire Elément> **Renommer** < Domaine; Décrire Domaine> **avec** <Domaine non Temporel; Décrire Domaine non Temporel> **Insérer branche** < Domaine; Décrire Domaine> **Insérer arc-choix entre** <Domaine ; Décrire Domaine> **et** <Elément ; Décrire Elément> **Insérer arc-choix entre** <Domaine ; Décrire Domaine> **et** <Domaine non Temporel; Décrire Domaine non Temporel>	< Elément; Décrire Elément > = " < Domaine; Décrire Domaine > <Domaine ; Décrire Domaine > = " < Domaine Temporel; Décrire Domaine Temporel> " < Domaine non Temporel; Décrire Domaine Non Temporel> < Domaine Temporel; Décrire Domaine Temporel> = " <Domaine Temporel Instant ; Décrire Domaine Temporel Instant> " <Domaine Temporel Période ; Décrire Domaine Temporel Période> " <Domaine Temporel Intervalle ; Décrire Domaine Temporel Intervalle> < Domaine Temporel Instant; Décrire Domaine Temporel Instant > = " <Domaine Temporel Instant-R; Décrire Domaine Temporel Instant-R> " <Domaine Temporel Instant-A ; Décrire Domaine Temporel Instant-A > < Domaine Temporel Intervalle; Décrire Domaine Temporel Intervalle > = " <Domaine Temporel Intervalle -R; Décrire Domaine Temporel Intervalle-R> " <Domaine Temporel Intervalle -A ; Décrire Domaine Temporel Intervalle -A >

Figure 183: Partie de la DÉMARCHE de O* après l'application du patron

On peut donc constater que la partie **DÉMARCHE** de la méthode O* a été étendue de manière à prendre en compte la construction de ce nouveau concept représentant les domaines temporels.

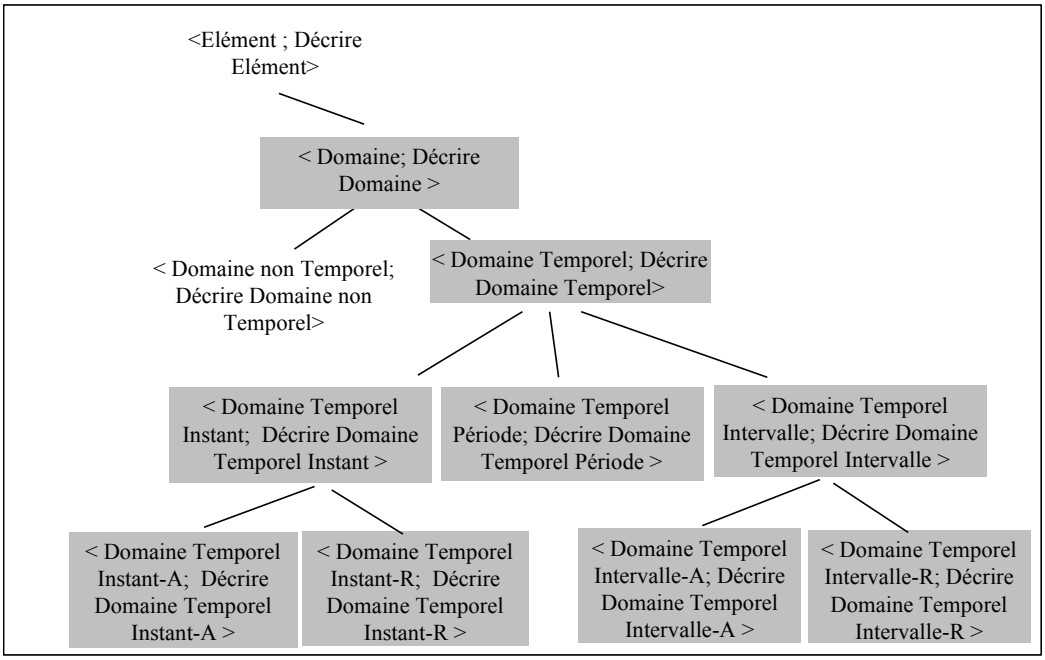


Figure 184: Partie de l'arbre de processus de la méthode O* étendue

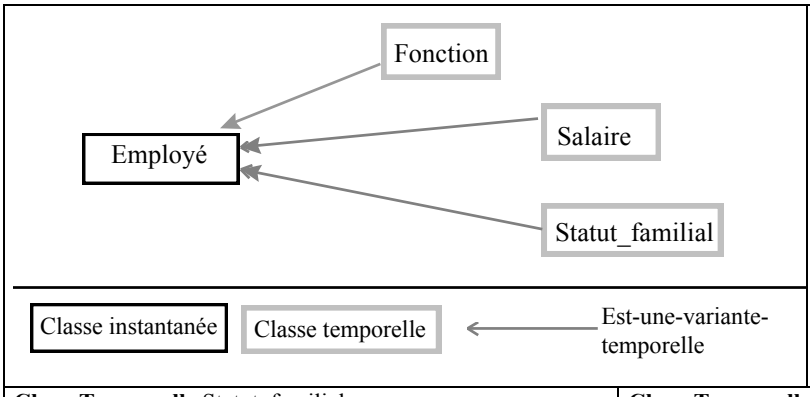
21.5 Insertion du concept d'historiques d'objets dans O*

21.5.1 Problème pris en compte

Des historiques sont parfois nécessaires dans certaines applications. Prenons l'exemple de l'évolution du salaire d'un employé. Si l'application garde seulement le dernier état de cet attribut, il sera

impossible d'étudier son évolution, alors que si l'application garde son historique, il sera très aisé de faire cette étude. De plus, la fonction de *Replay* est de plus en plus demandée pour définir la trace du processus décisionnel d'une organisation. Le SI doit donc fournir, pour chaque état d'un objet, quand il est/était vrai et quand il est/etait exploitable dans la base. De ce point de vue, les modèles OO devraient fournir des concepts spécifiques pour gérer l'historisation des objets avec les temps de validité et de transaction. Ces deux dimensions temporelles doivent être prises en compte dans le raisonnement de modélisation des données temporelles. Cette extension conduit à deux types de classes : les classes instantanées et les classes temporelles.

Dans le but de maintenir la consistance de la base de données, les objets temporels doivent être reliés à des objets non temporels. Ceux-ci sont appelés des classes instantanées. Par exemple, la *fonction d'un employé* est une variante temporelle de la classe *Employé*. Dans l'orientation objet, il existe au moins une propriété qui ne change jamais : l'identité de l'objet. En outre, une classe instantanée peut avoir plusieurs variantes temporelles. Le lien d'une classe temporelle à sa classe instantanée s'appelle « est-une-variante-temporelle ». La Figure 185 illustre le fait qu'un employé a trois variations temporelles : son salaire, sa fonction et son statut familial.



ClasseInstantanée Employé
propriétés
Nom : Chaîne
Adresse : Chaîne
finClasse

ClasseTemporelle Function
Variante de Employé
dimensions temporelles Valide INTERVALLE <Gregorien,Mois>
propriétés
titre : Chaîne
finClasse

ClasseTemporelle Statut_familial
Variante de Employé
dimensions temporelles Valide INTERVALLE <Gregorien,Jour>
propriétés
status : FAMILYSTATUS
finClasse

ClasseTemporelle Salaire
Variante de Employé
dimensions temporelles Valide INTERVALLE <Gregorien,Mois>
propriétés
Montant : REAL
finClasse

Figure 185 : Exemples de classes temporelles.

Une ***classe instantanée*** est une classe dont les objets représentent l'état courant de leurs correspondants réels. En d'autres termes l'objet informationnel ne garde pas trace de son évolution dans le temps. Il ne conserve que son dernier état. Par conséquent, les mises à jour sur ce type d'objet entraînent une perte de l'état précédent au profit du nouvel état. Une classe peut être naturellement « instantanée » si les propriétés sont permanentes (elles n'évoluent pas au cours du temps) ou si la représentation des états successifs des objets est sans intérêt. Par exemple, la classe *commande* est une classe instantanée si les aspects d'une commande sont immuables. Cette classe n'est pas sensible au temps. Chaque fois qu'un objet d'une classe instantanée est modifié ou supprimé, le nouvel état de l'objet remplace l'ancien état. Rappelons que l'*état d'un objet* est l'ensemble des valeurs de ses propriétés, et que *l'identité d'un objet* est une propriété immuable qui permet de distinguer un objet indépendamment de son état.

Une classe temporelle est une classe gérant le temps. Elle permet d'intégrer le temps de validité et/ou de transaction dans sa définition. Par exemple, le *salaire d'un employé* est estampillé par la date de validité de son salaire.

La classe temporelle permet de grouper des propriétés qui évoluent en même temps et sont associées au(x) même(s) dimension(s) temporelle(s). Par exemple le *salaire* et la *fonction d'un employé* sont dans deux classes temporelles différentes puisqu'elles n'évoluent pas en même temps.

Nous intégrons, dans le concept de *classe temporelle*, la possibilité d'avoir une historisation ou un estampillage du dernier état.

La notion ***d'état estampillé*** permet d'associer, au dernier état d'un objet, une estampille temporelle qui est le temps de validité de cet état dans le monde réel et/ou le temps de mise à jour de cet état dans la base de données (temps de transaction). Nous parlons *d'état valide* lorsque l'estampille est le temps de validité, *état système* lorsque le temps associé est le temps de transaction et enfin l'*état* est dit *bitemporel* lorsque l'estampille comprend les temps de validité et de transaction.

La notion ***d'histoire*** permet de garder tous les états estampillés d'un objet afin de retracer son évolution au cours du temps. L'*histoire valide* d'un objet reflète son évolution par rapport au monde réel (temps de validité). *L'histoire système* d'un objet permet d'avoir l'évolution de ses mises à jour (temps de transaction). L'histoire est dite "bitemporelle" lorsqu'elle combine le point de vue du monde réel et celui de la représentation.

La classe temporelle est caractérisée par la dimension temporelle à gérer et les deux options: estampillage ou historisation. Le croisement de ces deux caractéristiques permet de connaître toutes les options possibles de gestion du temps associées à la classe temporelle. La Figure 186 synthétise les options possibles.

		Temps de transaction		
		néant	estampillage	historisation
temps de validité	néant	sans gestion du temps (classe instantanée)	dernier état système	histoire système
	estampillage	dernier état valide	dernier état bitemporel	histoire système et le dernier état valide
	historisation	histoire valide	histoire valide et le dernier état système	histoire bitemporelle

Figure 186 : Options possibles dans une classe temporelle

21.5.2 Choix du patron d'extension

L'insertion du concept permettant la représentation des historiques d'objets est illustrée sur la carte comme l'intention cible de douze sections différentes. Six de ces sections ont comme intention source *Démarrer*, les six autres *Etendre Proc[M] pour intégrer la construction des domaines temporels*. Cela signifie qu'il y a six stratégies possibles entre cette intention et notre intention cible ainsi que six stratégies possibles entre *Démarrer* et notre intention cible, ainsi que le visualise la figure suivante.

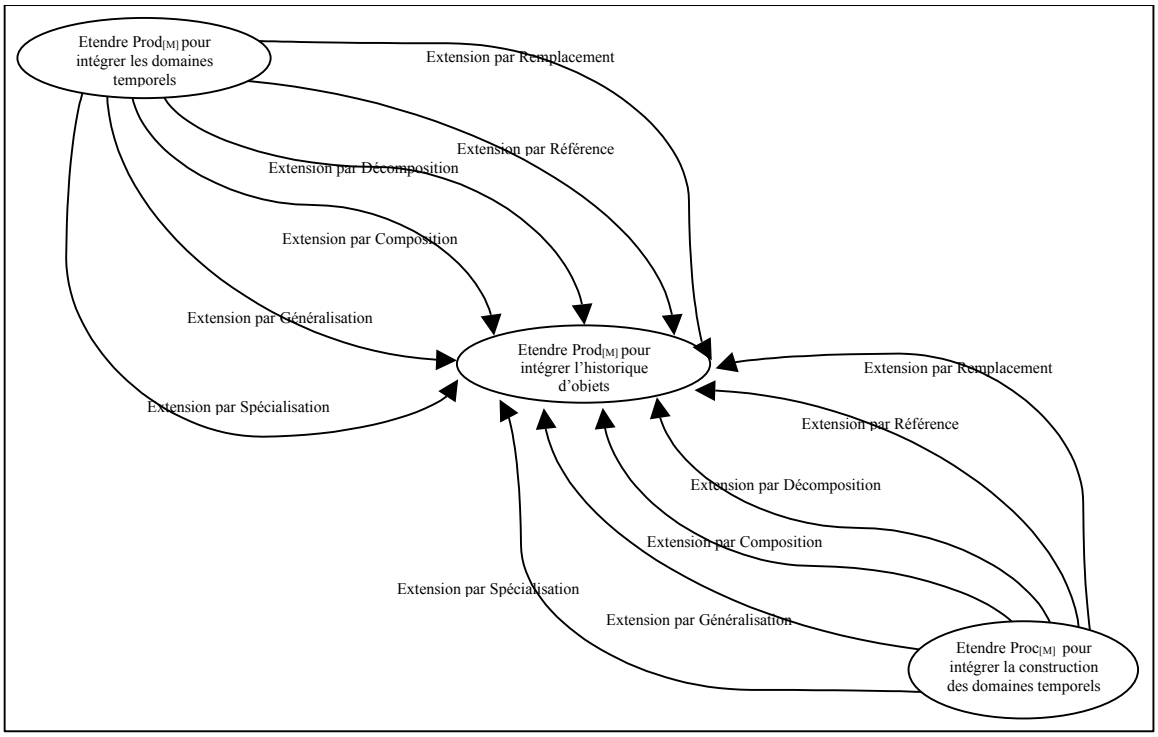


Figure 187: Sections de la carte d'extension d'une méthode aux applications temporelles ayant comme intention cible l'insertion d'historiques d'objets

La dernière intention réalisée dans notre chemin sur la carte d'extension étant celle d'*Etendre Proc[M] pour intégrer la construction des domaines temporels*, il nous faut choisir une stratégie dans l'ensemble des six stratégies ayant cette intention comme intention source. Le choix de cette stratégie est dépendant de la situation actuelle de O*.

Le modèle possède le concept de *Classe Objet* qui est une spécialisation du concept de *Classe*. Ce concept a trois spécialisations possibles : *Classe Objet*, *Classe Message* et *Classe Calendrier* (ce dernier ayant été intégré en lieu et place du concept de *Classe Horloge* lors de l'application du patron permettant l'insertion du concept de référentiel temporel).

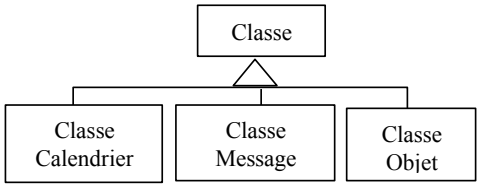
Description du **PRODUIT** avant extension	Représentation du **PRODUIT** avant extension
Classe = **Spécialisation** (Classe Calendrier, Classe Message, Classe Objet)	

Figure 188 : Partie du PRODUIT de O* avant l'application du patron

Le concept de *Classe d'Objet* peut être une spécialisation alternative au concept d'historiques d'objets. La stratégie choisie sera donc la stratégie d'extension du **PRODUIT** par GENERALISATION.

L'ingénieur de méthodes exécute donc le patron d'extension décrit dans l'annexe B et dont l'intention est d'*« Etendre, par GENERALISATION, le concept de Classe Objet pour intégrer le concept des historiques d'objets».*

21.5.3 Instanciation du patron d'extension à la méthode O*

L'interface du patron d'extension instancié à O* est la suivante.

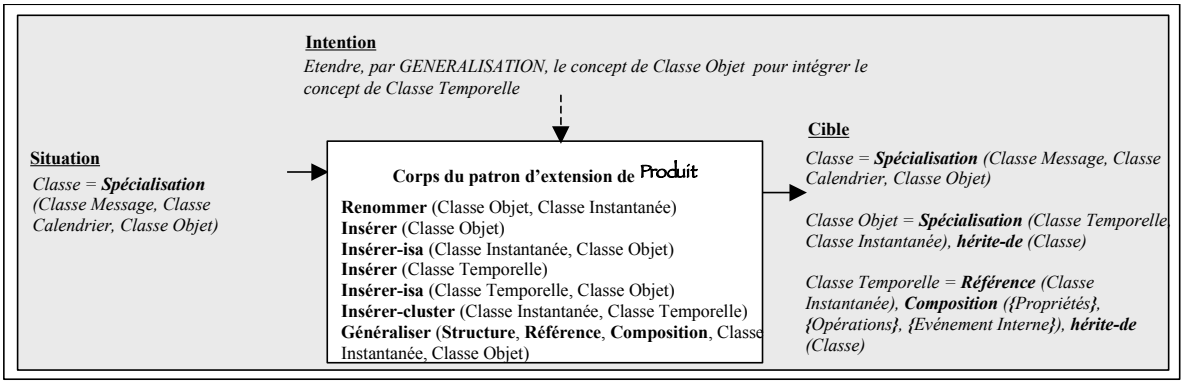


Figure 189: Application du patron d'extension de la méthode O* permettant de remplacer le concept des domaines temporels

Le corps du patron est donc composé de plusieurs opérateurs permettant de pratiquer cette modification pour conserver l'évolution des données de l'application. Ces opérateurs sont décrits dans la figure suivante.

Corps du Patron	Description du **PRODUIT** après extension
Renommer (Classe Objet, Classe Instantanée) **Insérer** (Classe Objet) **Insérer-isa** (Classe Instantanée, Classe Objet) **Insérer** (Classe Temporelle) **Insérer-isa** (Classe Temporelle, Classe Objet) **Insérer-cluster** (Classe Instantanée, Classe Temporelle) **Généraliser** (**Structure**, **Référence**, **Composition**, Classe Instantanée, Classe Objet)	Classe = **Spécialisation** (Classe Message, Classe Calendrier, Classe Objet) Classe Objet = **Spécialisation** (Classe Temporelle, Classe Instantanée), **hérite-de** (Classe) Classe Temporelle = **Référence** (Classe Instantanée), **Composition** ({Propriétés}, {Opérations}, {Evénement Interne}), **hérite-de** (Classe)

Figure 190: Partie du PRODUIT de O* après l'application du patron

On peut voir sur la Figure 191 que la partie **PRODUIT** de la méthode O* a été étendue pour intégrer une généralisation du concept de *Classe Objet* pour permettre à l'ingénieur de méthodes de définir le concept de *Classe Temporelle*, concept permettant de garder l'évolution dans le temps d'un ensemble de propriétés relatives à un objet d'une classe non temporelle (*Classe Instantanée*).

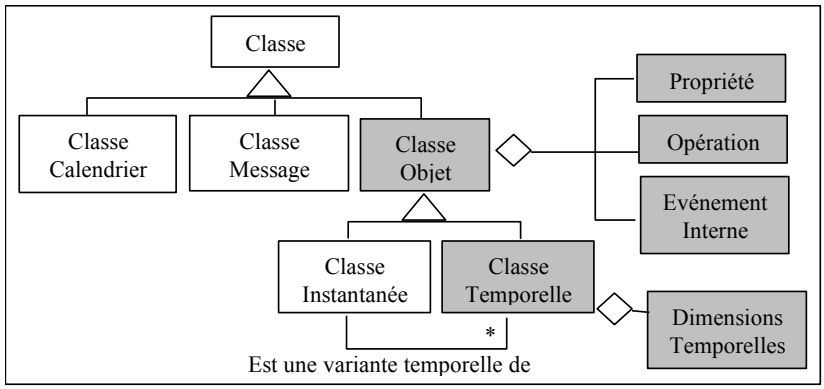


Figure 191: Partie du méta-modèle de la méthode O* étendue

On peut voir sur cette figure que le concept de *Classe Objet* est renommé puis généralisé pour permettre de lui donner la spécialisation alternative de *Classe Temporelle*. La composition du concept de *Classe Objet* reste au niveau du concept généralisé pour que les deux spécialisations puissent en hériter. De plus, le concept de *Classe Temporelle* définit également ce que l'on a appelé les *Dimensions Temporelles* (pour permettre l'historisation des objets avec soit un temps de validité, soit un temps de transaction, soit les deux). Ce concept est aussi relié au concept de *Classe Objet* par un lien de référence signifiant que toute instance de ce concept est une variante temporelle d'une classe n'évoluant pas dans le temps.

21.6 Insertion de la construction des historiques d'objets dans O*

21.6.1 Problème pris en compte

L'ingénieur de méthodes a intégré le concept d'historique d'objets dans la partie **PRODUIT** de la méthode d'origine et il souhaite maintenant étendre la partie **DÉMARCHE** de celle-ci pour pouvoir y intégrer la construction de ce concept.

21.6.2 Choix du patron d'extension

La construction du concept permettant de conserver l'historique d'objets est représentée dans la carte d'extension comme l'intention cible de trois sections dont l'intention source est *Démarrer*. Cela signifie qu'il existe trois stratégies différentes pour parvenir à cette intention, c.-à-d. trois patrons différents que l'on peut appliquer, comme le montre la figure suivante.

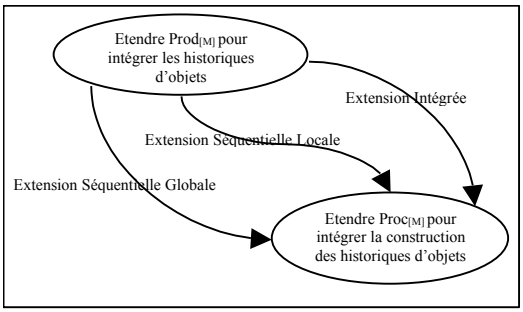


Figure 192: Sections de la carte d'extension d'une méthode aux applications temporelles ayant comme intention cible l'insertion de la construction des historiques d'objets

Le choix de la stratégie est dépendant de la situation en cours de la méthode d'origine, O*.

Le modèle de la méthode O* possède maintenant le concept de *Classe Temporelle* qui y a été intégré par la stratégie GENERALISATION. Cependant, la partie **DÉMARCHE** de O* n'a pas encore pris cette modification en compte.

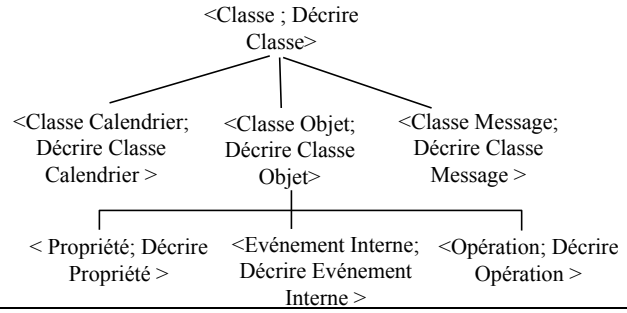
Description de la **DÉMARCHE** avant extension	Représentation de la **DÉMARCHE** avant extension
<Classe ; Décrire Classe> = " <Classe Objet ; Décrire Classe Objet> " <Classe Message; Décrire Classe Message> " <Classe Calendrier; Décrire Classe Calendrier> <Classe Objet; Décrire Classe Objet> = • < Propriété; Décrire Propriété> • < Evénement Interne; Décrire Evénement Interne> • < Opération; Décrire Opération>	

Figure 193 : Partie de la DÉMARCHE de O* avant l'application du patron

Différentes stratégies d'extension de la **DÉMARCHE** sont possibles après cette stratégie d'extension du **PRODUIT.** Comme l'ingénieur de méthodes a commencé son exécution de l'extension de la **DÉMARCHE** de O* par la stratégie INTEGREE, il est plus cohérent de continuer ainsi et d'intégrer toute modification de la **DÉMARCHE** de cette manière. De la même façon, nous avons privilégié le principe de la spécialisation par type par rapport à la spécialisation par état, la stratégie à appliquer pour cette extension spécifique sera donc celle de l'extension par GENERALISATION INTEGREE (par type).

L'ingénieur de méthodes exécute donc le patron d'extension décrit dans l'annexe B et dont l'intention est *« Etendre, par GENERALISATION INTEGREE (par type), la construction du concept de Classe Objet, pour y intégrer la construction des historiques d'objets ».*

21.6.3 Instanciation du patron d'extension à la méthode O*

L'interface du patron d'extension instancié à O* est la suivante.

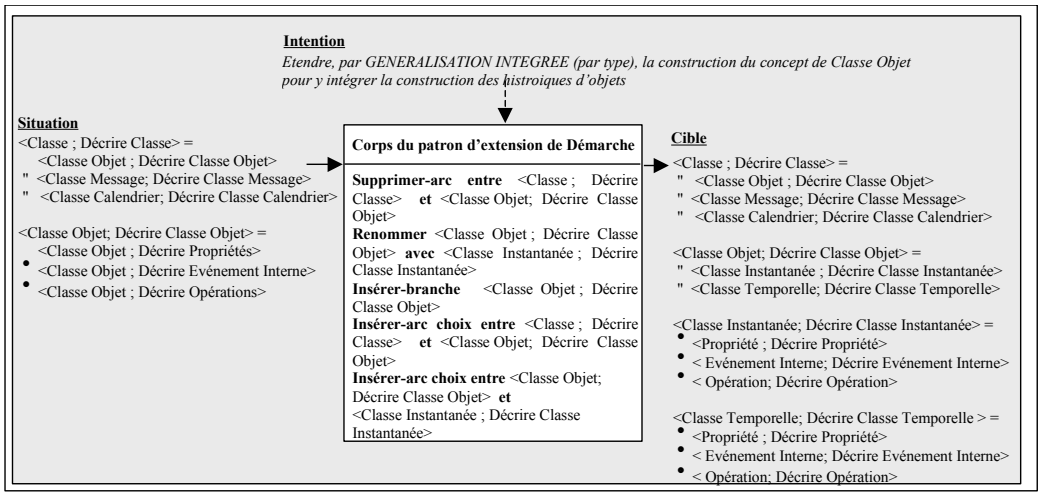


Figure 194: Application du patron d'extension de la méthode O* permettant d'intégrer la construction des historiques d'objets.

Le corps du patron est composé d'un ensemble d'opérateurs à exécuter sur la partie **DÉMARCHE** de la méthode O* pour permettre ces modifications.

Corps du Patron	Description de la **DÉMARCHE** après extension
Supprimer-arc entre <Classe ; Décrire Classe> **et** <Classe Objet; Décrire Classe Objet> **Renommer** <Classe Objet ; Décrire Classe Objet> **avec** <Classe Instantanée ; Décrire Classe Instantanée> **Insérer-branche** <Classe Objet ; Décrire Classe Objet> **Insérer-arc choix entre** <Classe ; Décrire Classe> **et** <Classe Objet; Décrire Classe Objet> **Insérer-arc choix entre** <Classe Objet; Décrire Classe Objet> **et** <Classe Instantanée ; Décrire Classe Instantanée>	<Classe ; Décrire Classe> = <Classe Objet ; Décrire Classe Objet> " <Classe Message; Décrire Classe Message> " <Classe Calendrier; Décrire Classe Calendrier> <Classe Objet; Décrire Classe Objet> = <Classe Instantanée ; Décrire Classe Instantanée> " <Classe Temporelle; Décrire Classe Temporelle> <Classe Instantanée; Décrire Classe Instantanée> = <Propriété ; Décrire Propriété> • < Evénement Interne; Décrire Evénement Interne> • < Opération; Décrire Opération> <Classe Temporelle; Décrire Classe Temporelle > = <Propriété ; Décrire Propriété> • < Evénement Interne; Décrire Evénement Interne> • < Opération; Décrire Opération> • < Classe Temporelle ; Définir Dimensions Temporelles>

Figure 195: Partie de la DÉMARCHE de O* après l'application du patron

On peut donc constater que la partie **DÉMARCHE** de la méthode O* a été étendue de manière à prendre en compte la construction de ce nouveau concept représentant les Classes Temporelles.

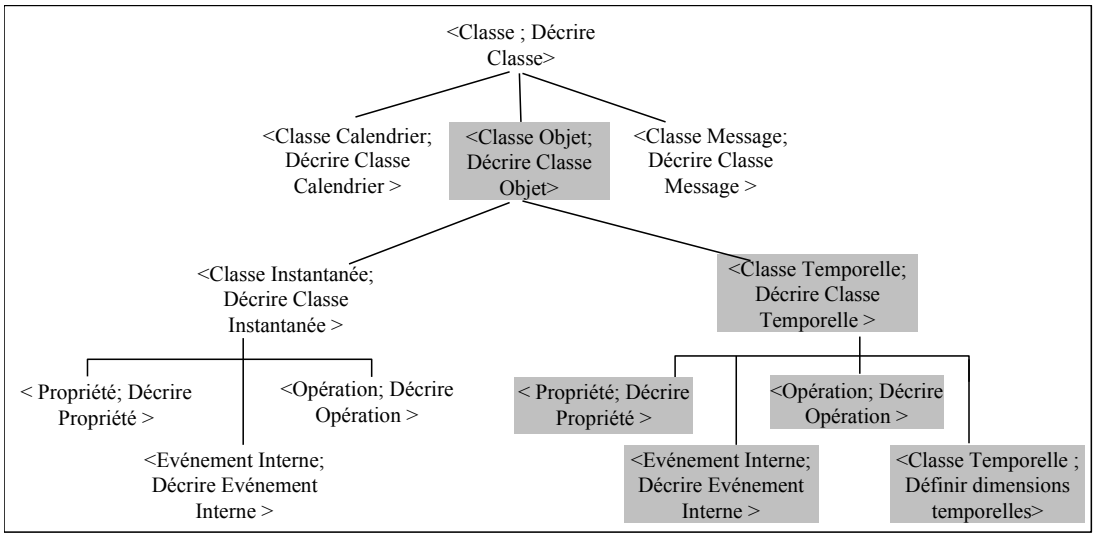


Figure 196: Partie de l'arbre de processus de la méthode O* étendue

21.7 Insertion du concept de contraintes temporelles dans O*

21.7.1 Problème pris en compte

Les contraintes d'intégrité associées à une base de données sont le plus souvent statiques, c'est à dire relatives au dernier état des données de la base. La gestion temporelle des données implique une prise en compte du temps dans l'expression des contraintes. Par exemple, on peut être amené à dire « *qu'un examinateur est affecté à un projet s'il n'a jamais été (ou n'est pas) employé par l'une des entreprise participant au consortium du projet* ».

Les contraintes temporelles ne peuvent s'appliquer que sur des classes temporelles. En effet, elles contraignent l'évolution des données contenues dans les historiques. Il est donc nécessaire d'avoir exécuté le patron de l'historisation des objets avant celui des contraintes temporelles.

Dans les classes d'objets, nous décrivons les contraintes que doivent vérifier les objets de cette classe [Rolland96]. Les contraintes sont principalement des contraintes : d'unicité, de cardinalité, d'héritage et d'attribut. Les contraintes d'unicité, d'héritage et d'attributs sont explicitement définies par le concepteur alors que les contraintes de cardinalité sont incluses dans les liens statiques. Nous allons d'abord rapidement rappeler la signification de chacune de ces contraintes et les définir dans un contexte temporel.

- Une contrainte d'unicité garantit l'unicité des objets d'une classe à partir de l'unicité des valeurs d'une propriété ou d'un ensemble de propriétés de la classe. Elle correspond à la contrainte de clé du relationnel. Par exemple, le numéro de sécurité sociale *NumSS* d'une personne est unique dans l'ensemble des personnes *Personne* : *unique (NumSS, Personne).*
- Une contrainte de cardinalité est définie dans chaque classe participant à une association ou à une agrégation. Par exemple, *une commande n'est passée* ***que par un seul*** *client*. La contrainte exprime une cardinalité instantanée. A un instant t donné combien de clients sont associés à une commande ? un et un seul client. La contrainte de cardinalité instantanée n'exprime pas ici que

durant la vie de la commande le client associé ne peut pas changer. L'aspect permanent ou variable de l'association n'est pas traité dans la cardinalité. Par exemple, *une voiture est la propriété, à un instant t, d'une seule personne*, n'empêche pas d'avoir plusieurs propriétaires différents durant la vie de la voiture.

- Une contrainte d'héritage s'exprime entre une classe généralisée et un ensemble de classes spécialisées. Les contraintes sont les contraintes de disjonction, de couverture et de partition. Les classes H*omme* et F*emme* sont des spécialisations disjointes de la classe *Personne*; c'est à dire qu'une personne ne peut pas être à la fois un homme et une femme. De plus les classes H*omme* et F*emme* couvrent la classe *Personne,* c.-à-d. une personne est obligatoirement une femme ou un homme. Enfin si les classes spécialisées combinent la contrainte de disjonction et la contrainte de couverture; elles constituent une partition. Par conséquent, les classes H*omme* et F*emme* forment une partition de la classe *Personne*.
- Une contrainte d'attribut est une contrainte qui s'applique sur une propriété statique d'un objet. Par exemple, le *poids du patient ne peut pas excéder 200 kilos* ou la *date de naissance du patient ne peut pas être antérieure à 1800* sont des exemples de contraintes d'attribut.

On propose une classification des contraintes qui combine les types classiques et ceux relatifs au temps. Cette classification est utile pour déterminer l'impact d'une contrainte locale à un objet ou globale pour un ensemble d'objets ainsi qu'à un instant t ou pour un ensemble d'états à des instants t différents.

Dans les SGBD actifs orientés objets comme ODE [Gehani91], les contraintes sont classées en deux catégories : les contraintes intra-objet et les contraintes inter-objet. Une contrainte *intra-objet* permet de définir l'intégrité localement à un objet par opposition à une contrainte *inter-objet* qui détermine l'intégrité globalement à un ensemble d'objets. L'application de cette classification aux cinq contraintes précédentes conduit au résultat suivant :

Contraintes	intra-objet	inter-objet
unicité		X
cardinalité		X
héritage		X
attribut	X[(1)]	X[(2)]

La deuxième classification est inspirée de celle proposée par Böhlen dans « *Classifications of temporal constraints* »; elle distingue les contraintes *intra-temps* et *inter-temps*.

Une contrainte *intra-temps* se définit localement à un instant t. Chaque instant t doit vérifier la contrainte. La sémantique est différente selon la dimension temporelle utilisée dans la contrainte.

[(1)] contraintes d'attribut n'accédant à aucun autre objet dans sa définition. Exemple, le poids du patient ne peut pas excéder 200 kilos.
[(2)] contraintes d'attribut accédant à d'autres objets dans sa définition. Exemple, la date de visite ne doit pas excéder d'un mois la visite précédente.

- Une contrainte intra-temps associée au temps de validité s'applique à l'état du monde réel à un instant t. Par exemple, l'affectation d'un employé à un projet doit exister pour la semaine de la charge hebdomadaire déclarée par cet employé pour ce projet.
- Une contrainte intra-temps associée au temps de transaction s'applique à l'état de la base de données à un instant t. Par exemple, si le nom d'une société est stocké dans une histoire<TT>, l'unicité du nom pour chaque état de la base de données est vérifiée aux temps de transaction.
- Une contrainte intra-temps associée à une estampille bi-temporelle s'applique à l'état du monde réel à un instant v stocké dans l'état de la base de données à un instant t. Par exemple, si le poids du patient est dans une histoire<VT,TT>, la contrainte « le poids ne doit pas excéder 200 kilos » doit se vérifier pour chaque couple <VT,TT> .

Une contrainte *inter-temps* se définit en utilisant des informations à des temps différents. Chaque instant t doit vérifier la contrainte. La sémantique change selon la dimension temporelle.

- Une contrainte inter-temps associée au temps de validité définit l'intégrité pour un ensemble d'états du monde réel. Par exemple, *une personne ne peut pas être embauchée plusieurs fois par la même compagnie* nécessite le parcours de la variation temporelle « employé » d'une personne.
- Une contrainte inter-temps associée au temps de transaction définit l'intégrité pour un ensemble d'états de la base de données. Par exemple, si le nom d'une personne est stocké dans une histoire<TT>, la contrainte « *une personne ne peut pas changer de nom plus de trois fois »* requiert l'accès à plusieurs temps de transaction.
- Une contrainte inter-temps associée à une estampille bi-temporelle s'applique à plusieurs états bi-temporels. Par exemple, si le poids du patient est dans une histoire<Bi>, la contrainte « *le poids à un instant de validité donné ne doit pas être corrigé plus de 3 fois »* utilise plusieurs états ayant une estampille bi-temporelle.

Les contraintes intra-temps sont faciles à vérifier puisqu'elles s'appliquent à l'état du monde réel ou l'état courant de la représentation que l'utilisateur vient de modifier. Par contre, les contraintes inter-temps sont plus difficiles à vérifier car elles nécessitent un accès à tous les états du monde réel que l'on connaît. La classification proposée combine ces deux modes de typage.

21.7.2 Choix du patron d'extension

L'insertion du concept permettant la représentation des contraintes temporelles est illustrée sur la carte comme l'intention cible de douze sections différentes. Six de ces sections ont comme intention source *Démarrer*, les six autres *Etendre Proc[M] pour intégrer la construction des historiques d'objets*. Cela signifie qu'il y a six stratégies possibles entre cette intention et notre intention cible ainsi que six stratégies possibles entre *Démarrer* et notre intention cible, comme le montre la figure suivante.

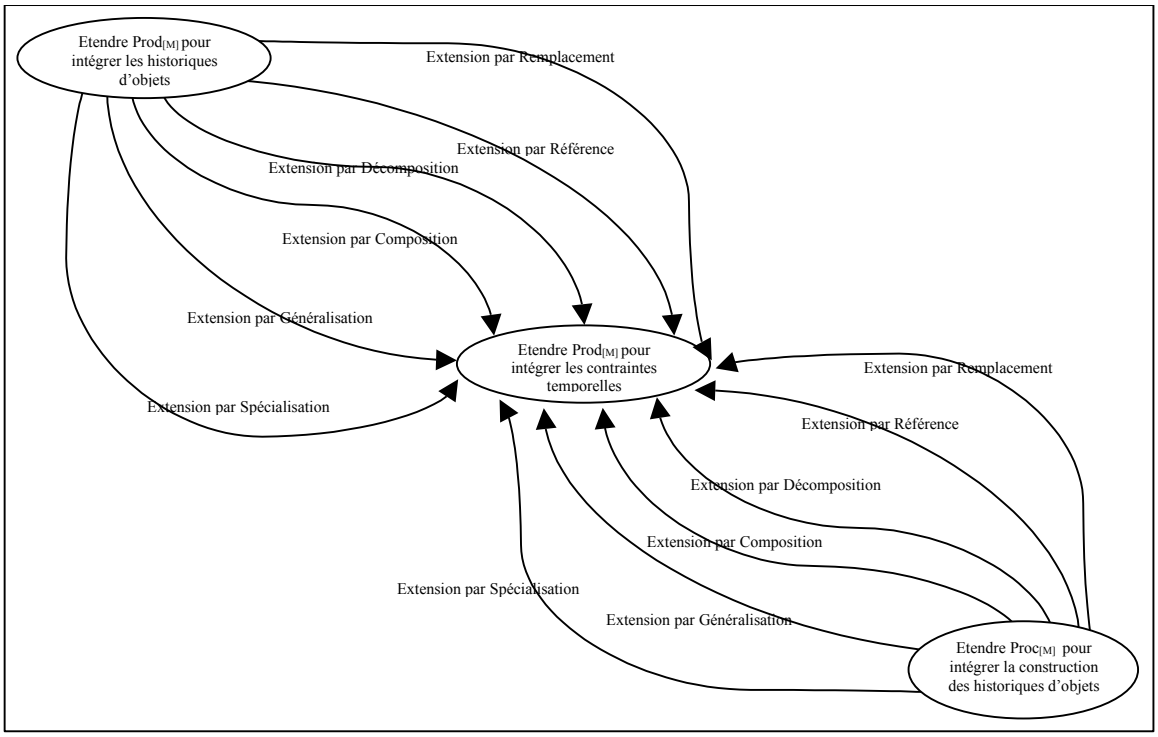


Figure 197: Sections de la carte d'extension d'une méthode aux applications temporelles ayant comme intention cible l'insertion des contraintes temporelles

La dernière intention réalisée dans notre chemin sur la carte d'extension étant celle d'*Etendre Proc[M] pour intégrer la construction des historiques d'objets*, il nous faut choisir une stratégie dans l'ensemble de celles ayant cette intention comme intention source. Le choix de cette stratégie est dépendant de la situation en cours de la méthode d'origine, ici O*.

Le modèle possède le concept de *Classe Temporelle* qui pourrait être un concept composé du concept de contrainte temporelle.

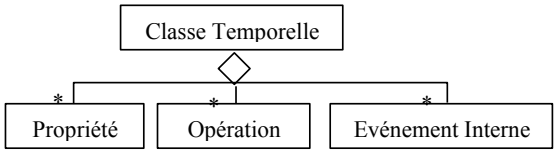
Description du **PRODUIT** avant extension	Représentation du **PRODUIT** avant extension
Classe Temporelle = **Composition** ({Propriété}, {Opération}, {Evénement Interne})	

Figure 198 : Partie du PRODUIT de O* avant l'application du patron

L'ingénieur de méthodes exécute donc le patron d'extension décrit dans l'annexe B et dont l'intention est d'*« Etendre, par COMPOSITION, le concept de Classe Objet pour intégrer le concept des contraintes temporelles».*

21.7.3 Instanciation du patron d'extension à la méthode O*

L'interface du patron d'extension instancié pour la méthode O* est la suivante.

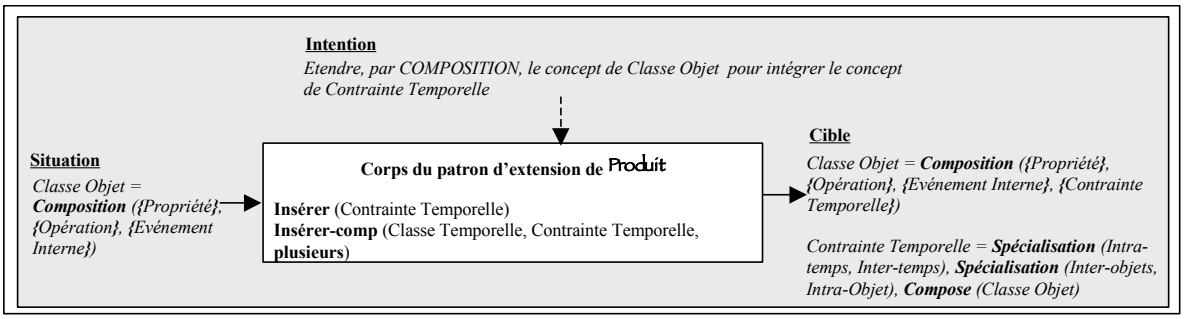


Figure 199: Application du patron d'extension de la méthode O* permettant de remplacer le concept des domaines temporels

La classification des contraintes intégrée par l'extension temporelle est la suivante.

- La contrainte de cardinalité s'applique aux références et aux compositions. Quel que soit le type de classe participant à la référence, la sémantique de la cardinalité ne change pas; elle est considérée comme instantanée, le temps à considérer ici est le temps de validité.
- La contrainte d'unicité s'applique aux classes instantanées et aux classes temporelles.
  - La contrainte ***d'unicité non temporelle*** se définit sur une collection non temporelle. Par exemple, le numéro de sécurité sociale doit être unique dans l'ensemble des personnes.
  - La contrainte ***d'unicité temporelle locale (intra-objet)*** s'applique à une collection temporelle d'une même classe. Par exemple, le propriétaire d'une voiture est unique dans l'historique des propriétaires de toutes les voitures (une personne ne peut pas posséder la même voiture plusieurs fois).
  - La contrainte ***d'unicité temporelle globale(inter-objet)*** s'applique à une collection temporelle intégrant plusieurs classes. Par exemple, le propriétaire d'une voiture est unique à un instant t dans l'historique des propriétaires de toutes les voitures (à un instant t une personne ne possède qu'une voiture).
- Les contraintes d'héritage ne changent pas; elles sont intemporelles.
- Les contraintes de propriété s'appliquent sur des classes instantanées et des classes temporelles. La catégorie de la contrainte dépend des objets participant à sa définition. Le Tableau 1 donne un exemple de chacune des catégories.

	intra-objet	inter-objet
sans temps	la date de naissance d'une personne est postérieure à l'année 1800.	La date de visite2 < la date de visite1 + 1,5 mois.
intra-vt	pour chaque état du monde réel, l'effort déclaré par un employé pour un projet et pour une semaine ne peut excéder 60 heures.	L'effort déclaré par un employé pour une semaine doit être égal à la somme des efforts qu'il a déclaré par projets pour cette même semaine.
intra-tt	pour chaque état base de données, le poids d'un patient (classe temporelle avec Histoire<TT>) est inférieur à 200 kilos.	Pour chaque état base de données, l'affectation d'un employé a un projet est possible si ils appartiennent au même département.
intra-Bi	pour chaque état de la classe temporelle effortProjet (histoire<VT,TT>), l'effort déclaré par un employé pour un projet et une semaine doit être inférieur à 60.	Pour chaque état de la classe temporelle, effortTotal (Histoire<VT, TT>), l'attribut effort total déclaré par un employé pour une semaine doit être égal à la somme des efforts déclarés par projet pour la même semaine.
inter-vt	Le salaire d'un employé (classe temporelle avec une Histoire<VT>) ne peut pas varier plus de 3 fois dans la même année.	le salaire d'un employé (classe temporelle avec une Histoire<VT>) ne peut pas avoir une augmentation supérieure à la limite fixée par département pour la même période.
inter-Bi	le poids d'un patient (classe temporelle avec une Histoire<VT, TT>) pour un temps de validité donné ne peut pas être corrigé plus de trois fois.	un examinateur est affecté à un projet (classe temporelle avec une histoire<VT,TT>) s'il n'a jamais été (ou n'est pas) employé par l'une des entreprises participant au consortium du projet ».

Tableau 1 : Exemples de contraintes de propriétés.

Cette classification des contraintes selon les deux canevas intra/inter objet/temps est utile pour la compréhension des contraintes que l'on formule mais aussi pour la phase logique où les contraintes devront être traduites dans le langage spécifique d'un SGBD temporel et orienté objet.

Prenons l'exemple (Figure 200) d'une gestion de projet où les efforts d'un employé pour une semaine et pour un projet sont indiqués par l'employé dans la classe temporelle A*ffectationHebdomadaire* alors que l'effort dans sa globalité pour un projet et pour une semaine est déclaré par le chef de projet dans la classe temporelle *EffortRéalisé*. L'effort réalisé par un employé ne peut pas excéder 40 heures et l'effort hebdomadaire déclaré par le chef de projet doit être égal à la somme des efforts réalisés par l'ensemble des employés affectés à ce projet pour cette semaine. Une contrainte est décrite par le texte annoncé par le mot clé *assertion* et sa catégorie par rapport aux classifications (intra/inter objet/temps).

Classe temporelle Effort-Réalisé **Variante de** Projet **Dimensions temporelles** Valide INTERVALLE <Grégorien, Semaine> **Propriétés** quantité : entier **Contraintes** assertion (*quantité est égale à la somme des quantités déclarées dans Affectation-Hebdomadaire pour ce projet et cette semaine*), inter-objet, intra-temps **Opérations** créer ... **FinClasse**	**Classe temporelle A**ffectation-Hebdomadaire **Variante de** Employé **Dimensions temporelles** Valide INTERVALLE <Grégorien, Semaine> **Propriétés** quantité : entier pour : ref (projet) **Contraintes** assertion (*quantité< 40 heures*), intra-objet, intra-temps **Opérations** créer ... **FinClasse**

Figure 200: Exemples de contraintes

Le corps de ce patron est composé de plusieurs opérateurs de modification. Ceux-ci sont décrits à la figure suivante.

Corps du Patron	Description du **PRODUIT** après extension
Insérer (Contrainte Temporelle) **Insérer-comp** (Classe Temporelle, Contrainte Temporelle, **plusieurs**)	Classe Temporelle = **Composition** ({Propriété}, {Opération}, {Evénement Interne},{Contrainte Temporelle}) Contrainte Temporelle = **Spécialisation** (Intra-temps, Inter-temps), **Spécialisation** (Inter-objets, Intra-Objet), **Compose** (Classe Objet)

Figure 201: Partie du PRODUIT de O* après l'application du patron

On peut voir sur la Figure 202 que la partie **PRODUIT** de la méthode O* a été étendue pour intégrer une composition du concept de Classe Objet avec les contraintes temporelles.

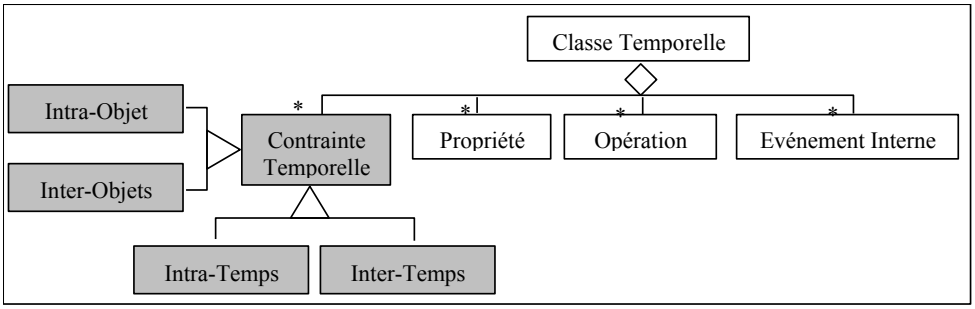


Figure 202: Partie du méta-modèle de la méthode O* étendue

On peut constater sur cette figure que l'intégration du concept de *Contrainte Temporelle* permet d'intégrer également la classification associée de celui-ci.

21.8 Insertion de la construction des contraintes temporelles dans O*

21.8.1 Problème pris en compte

L'ingénieur de méthodes a intégré le concept de contrainte temporelle dans la partie **PRODUIT** de la méthode d'origine et il souhaite maintenant étendre la partie **DÉMARCHE** de celle-ci pour pouvoir y ajouter la description de ce concept.

21.8.2 Choix du patron d'extension

La description du concept permettant de contraindre les données des classes temporelles est représentée dans la carte d'extension comme l'intention cible de trois sections dont l'intention source est *Démarrer*. Cela signifie qu'il existe trois stratégies différentes pour parvenir à cette intention, c.-à-d. trois patrons différents que l'on peut appliquer, comme le montre la figure suivante.

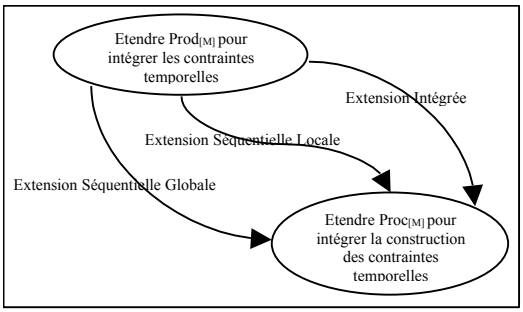


Figure 203: Sections de la carte d'extension d'une méthode aux applications temporelles ayant comme intention cible l'insertion de la construction des contraintes temporelles

Le modèle de la méthode O* possède maintenant le concept de *Contrainte Temporelle* qui y a été intégré par la stratégie COMPOSITION du concept de *Classe Objet*. Cependant, la partie **DÉMARCHE** de O* n'a pas encore pris cette modification en compte.

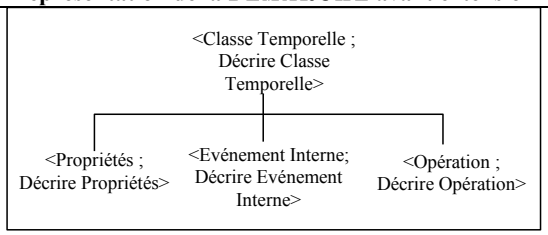
Description de la **DÉMARCHE** avant extension	Représentation de la **DÉMARCHE** avant extension
<Classe Temporelle ; Décrire Classe Temporelle> = • <Propriétés ; Décrire Propriétés> • <Opération ; Décrire Opération> • <Evénement Interne; Décrire Evénement Interne >	

Figure 204 : Partie de la DÉMARCHE de O* avant l'application du patron

Différentes stratégie d'extension de la **DÉMARCHE** sont possibles après cette stratégie d'extension du **PRODUIT.** Comme l'ingénieur de méthodes a commencé son exécution de l'extension de la **DÉMARCHE** de O* par la stratégie INTEGREE, il paraît cohérent de continuer ainsi et d'intégrer toute modification de la **DÉMARCHE** de cette manière.

L'ingénieur de méthodes exécute donc le patron d'extension décrit dans l'annexe B et dont l'intention est *« Etendre, par COMPOSITION INTEGREE, la construction du cocnept de Classe Objet pour intégrer la construction des historiques d'objets ».*

21.8.3 Instanciation du patron d'extension à la méthode O*

L'interface du patron d'extension instancié pour la méthode O* est la suivante.

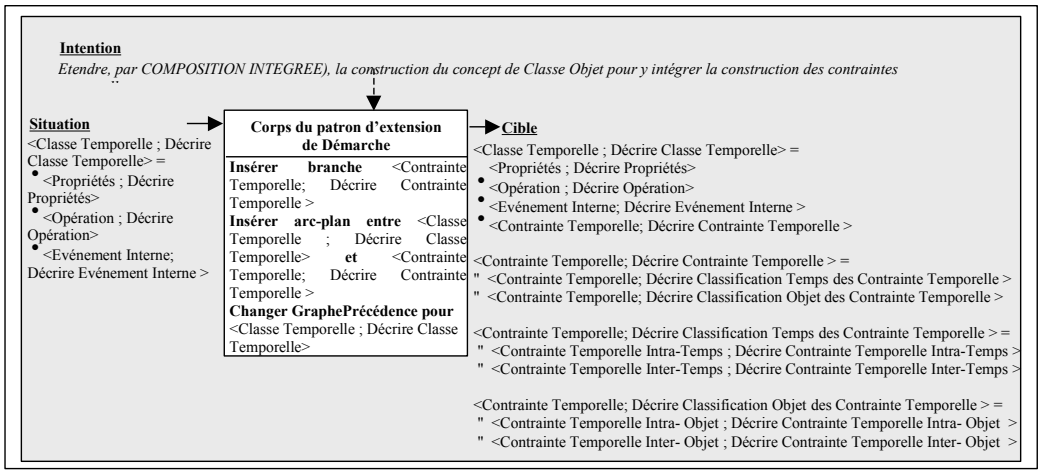


Figure 205: Application du patron d'extension de la méthode O* permettant d'intégrer la construction des contraintes temporelles.

Le corps de ce patron est composé d'un ensemble d'opérateurs permettant d'effectuer l'extension de O*. Ces opérateurs sont décrits dans la figure suivante.

Corps du Patron	Description de la **DÉMARCHE** après extension
Insérer branche <Contrainte Temporelle; Décrire Contrainte Temporelle > **Insérer arc-plan entre** <Classe Temporelle ; Décrire Classe Temporelle> **et** <Contrainte Temporelle; Décrire Contrainte Temporelle > **Changer GraphePrécédence pour** <Classe Temporelle ; Décrire Classe Temporelle>	<Classe Temporelle ; Décrire Classe Temporelle> = • <Propriétés ; Décrire Propriétés> • <Opération ; Décrire Opération> • <Evénement Interne; Décrire Evénement Interne > • <Contrainte Temporelle; Décrire Contrainte Temporelle > <Contrainte Temporelle; Décrire Contrainte Temporelle > = " <Contrainte Temporelle; Décrire Classification Temps des Contrainte Temporelle > " <Contrainte Temporelle; Décrire Classification Objet des Contrainte Temporelle > <Contrainte Temporelle; Décrire Classification Temps des Contrainte Temporelle > = " <Contrainte Temporelle Intra-Temps ; Décrire Contrainte Temporelle Intra-Temps > " <Contrainte Temporelle Inter-Temps ; Décrire Contrainte Temporelle Inter-Temps > <Contrainte Temporelle; Décrire Classification Objet des Contrainte Temporelle > = " <Contrainte Temporelle Intra- Objet ; Décrire Contrainte Temporelle Intra- Objet > " <Contrainte Temporelle Inter- Objet ; Décrire Contrainte Temporelle Inter- Objet >

Figure 206: Partie de la DÉMARCHE de O* après l'application du patron

On peut donc constater que la partie **DÉMARCHE** de la méthode O* a été étendue de manière à prendre en compte la construction de ce nouveau concept représentant les Contraintes Temporelles.

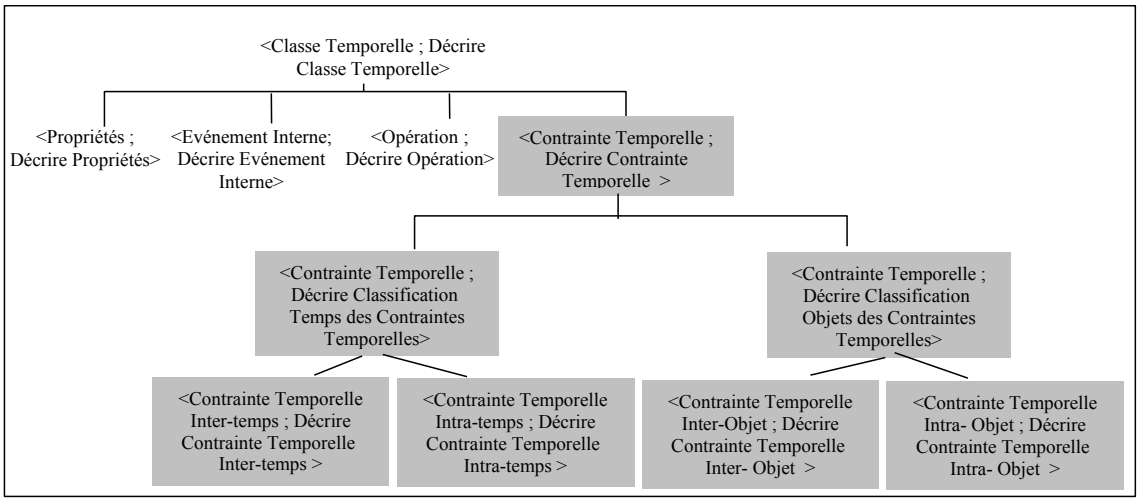


Figure 207: Partie de l'arbre de processus de la méthode O* étendue

21.9 Insertion du concept d'estampillage d'objets dans O*

21.9.1 Problème pris en compte

L'ingénieur d'application peut vouloir estampiller des données avec différentes sémantiques temporelles. Par exemple, l'attribut *order-date* représente la date de la commande du client : la vraie date de la réalité, alors que l'attribut *last-modification* représente l'heure à laquelle la commande a été enregistrée dans la base de données. Le premier est appelé *temps de validité* et le second *temps de transaction*.

21.9.2 Choix du patron d'extension

L'insertion du concept permettant l'estampillage des objets dans une méthode objet est représentée dans la carte d'extension comme l'intention cible de six sections spécifiques dont l'intention source est *Démarrer*. La diversité de ces sections ayant même intention source et même intention cible signifie qu'il existe autant de stratégies différentes pour atteindre cette intention, ainsi que le montre la Figure 208.

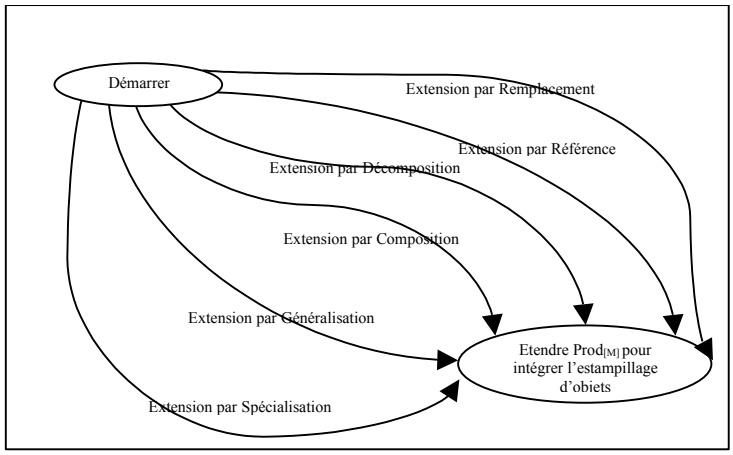


Figure 208: Sections de la carte d'extension d'une méthode aux applications temporelles ayant comme intention cible l'insertion des estampillages d'objets

Le choix de la stratégie à appliquer, et donc de la section à sélectionner (et par là même du patron à exécuter) dépend de la situation courante de la méthode d'origine, O*.

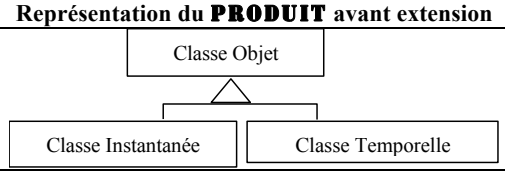
Description du **PRODUIT** avant extension	Représentation du **PRODUIT** avant extension
Classe Objet = **Spécialisation** (Classe Instantanée, Classe Temporelle)	

Figure 209 : Partie du PRODUIT de O* avant l'application du patron

Le concept spécifique de *Classe Estampillée* (que ce soit par le temps de transaction ou par le temps de validité) peut être considéré comme un concept alternatif à celui de *Classe Instantanée* ou *Classe Temporelle*, autrement dit, comme une nouvelle spécialisation de celui de *Classe Objet*. La stratégie à utiliser dans ce cas est donc celle de l'extension par SPECIALISATION

L'ingénieur de méthodes exécute donc le patron d'extension décrit dans l'annexe B et dont l'intention est d'*« Etendre, par SPECIALISATION, le concept de Classe Objet pour intégrer le concept d'estampillage d'objets».*

21.9.3 Instanciation du patron d'extension à la méthode O*

L'interface du patron d'extension instancié pour la méthode O* est la suivante.

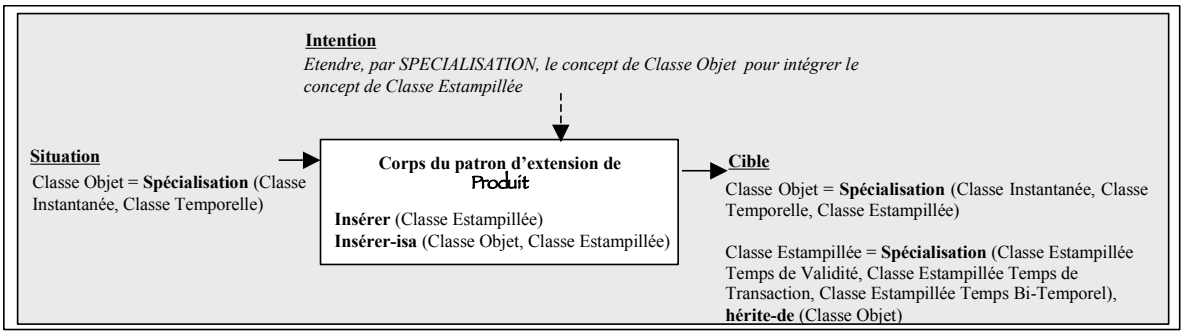


Figure 210: Application du patron d'extension de la méthode O* permettant de remplacer le concept d'estampillage d'objets

Le résultat de l'application de ce patron est la définition d'une hiérarchie de classes estampillées. Il y a différents types d'estampilles : une avec le temps de validité, une avec le temps de transaction et une avec les deux temps (estampille bi-temporelle). Comme le montre la Figure 212, chacune de ces estampilles est représentée par une classe.

Le temps de validité d'un fait est le temps où il est vrai dans la réalité. Par exemple, *James.B.* a été employé par la *Expert Company* à partir du *1995-02-01*. Cette définition autorise également à représenter de la même manière un instant ou un intervalle lorsque le fait est valide.

Le temps de transaction d'un fait de base de données est le temps courant d'un fait dans la base de données. Par exemple, l'emploi de *James.B.* par la *Expert company* est disponible dans la base de données à partir du *1995-01-10 14:56:50*.

Prenons l'exemple de la classe *Order* estampillée par les temps <VT, TT>. L'ingénieur veut savoir quand la commande a été faite par le client (temps réel) et également quand elle a été implantée dans la base de données (cela peut l'aider par exemple à gérer les délais possibles entre ces temps). Cette classe est donc estampillée par le temps de validité et par le temps de transaction, elle est appelée classe estampillée bi-temporelle.

Corps du Patron	Description du **PRODUIT** après extension
Insérer (Classe Estampillée) **Insérer-isa** (Classe Objet, Classe Estampillée)	Classe Objet = **Spécialisation** (Classe Instantanée, Classe Temporelle, Classe Estampillée) Classe Estampillée = **Spécialisation** (Classe Estampillée Temps de Validité, Classe Estampillée Temps de Transaction, Classe Estampillée Temps Bi-Temporel), **hérite-de** (Classe Objet)

Figure 211: Partie du PRODUIT de O* après l'application du patron

On peut voir sur la Figure 212 que la partie **PRODUIT** de la méthode O* a été étendue pour intégrer le concept de *Classe Estampillée* permettant d'intégrer également toutes ses spécialisations selon le type de l'estampille associé à la classe.

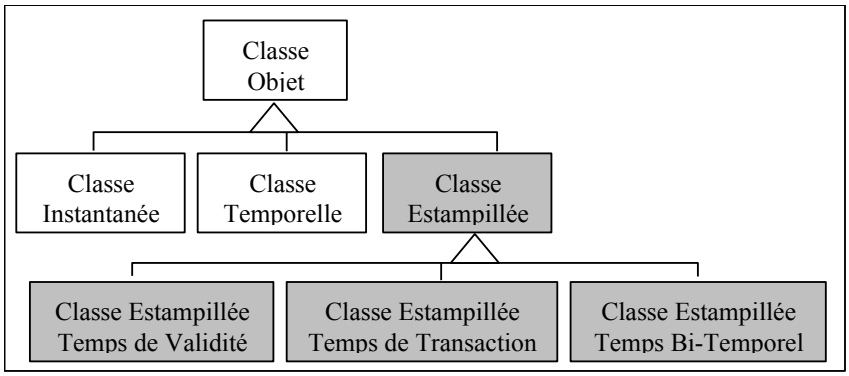


Figure 212: Partie du méta-modèle de la méthode O* étendue

21.10 Insertion de la construction des estampillages d'objets dans O*

21.10.1 Problème pris en compte

L'ingénieur de méthodes a intégré le concept d'estampillage d'objets dans la partie **PRODUIT** de la méthode d'origine et il souhaite maintenant étendre la partie **DÉMARCHE** de celle-ci pour pouvoir y intégrer la construction de ce concept.

21.10.2 Choix du patron d'extension

La construction du concept permettant de décrire l'estampillage des objets est représentée dans la carte d'extension comme l'intention cible de trois sections dont l'intention source est *Démarrer*. Cela signifie qu'il existe trois stratégies différentes pour parvenir à l'intention qui nous intéresse, c'est à dire trois patrons différents que l'on peut appliquer, comme le décrit la figure suivante.

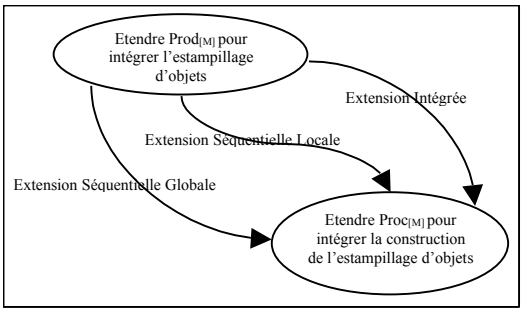


Figure 213: Sections de la carte d'extension d'une méthode aux applications temporelles ayant comme intention cible l'insertion de la construction de l'estampillage d'objets

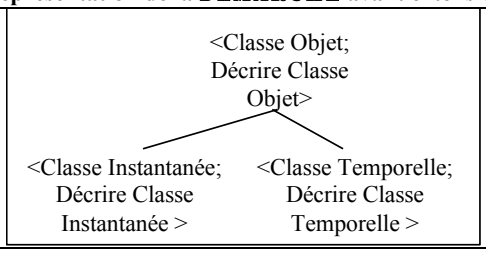
Description de la **DÉMARCHE** avant extension	Représentation de la **DÉMARCHE** avant extension
<Classe Objet : Décrire Classe Objet> = <Classe Instantanée ; Décrire Classe Instantanée> " <Classe Temporelle ; Décrire Classe Temporelle>	

Figure 214 : Partie de la DÉMARCHE de O* avant l'application du patron

Différentes stratégie d'extension de la **DÉMARCHE** sont possibles après cette stratégie d'extension du **PRODUIT.** De la même manière que pour les patrons précédents, étant donné que l'ingénieur de méthodes a commencé son exécution de l'extension de la **DÉMARCHE** de O* par la stratégie INTEGREE, il est cohérent de continuer ainsi et d'intégrer toute modification de la **DÉMARCHE** de cette manière.

L'ingénieur de méthodes exécute donc le patron d'extension décrit dans l'annexe B et dont l'intention est *« Etendre, par SPECIALISATION INTEGREE (par type), la construction du concept de Classe Objet pour intégrer la construction des objets estampillés ».*

21.10.3 Instanciation du patron d'extension à la méthode O*

L'interface du patron d'extension instancié pour la méthode O* est la suivante.

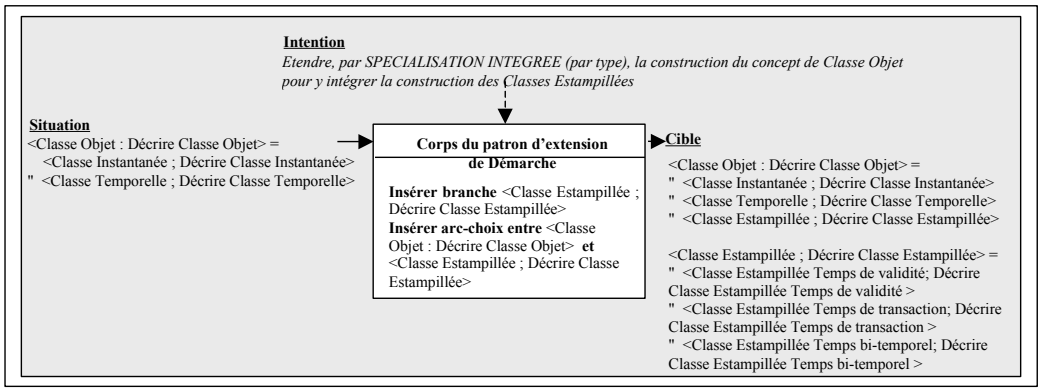


Figure 215: Application du patron d'extension de la méthode O* permettant d'intégrer la construction des contraintes temporelles.

Le corps de ce patron est composé de deux opérateurs permettant d'effectuer cette extension de la méthode O*.

Corps du Patron	Description de la **DÉMARCHE** après extension
Insérer branche <Classe Estampillée ; Décrire Classe Estampillée> **Insérer arc-choix entre** <Classe Objet : Décrire Classe Objet> **et** <Classe Estampillée ; Décrire Classe Estampillée>	<Classe Objet : Décrire Classe Objet> = " <Classe Instantanée ; Décrire Classe Instantanée> " <Classe Temporelle ; Décrire Classe Temporelle> " <Classe Estampillée ; Décrire Classe Estampillée> <Classe Estampillée ; Décrire Classe Estampillée> = " <Classe Estampillée Temps de validité; Décrire Classe Estampillée Temps de validité > " <Classe Estampillée Temps de transaction; Décrire Classe Estampillée Temps de transaction > " <Classe Estampillée Temps bi-temporel; Décrire Classe Estampillée Temps bi-temporel >

Figure 216: Partie de la DÉMARCHE de O* après l'application du patron

On peut donc constater que la partie **DÉMARCHE** de la méthode O* a été étendue de manière à prendre en compte la construction de ce nouveau concept représentant l'estampillage d'objets.

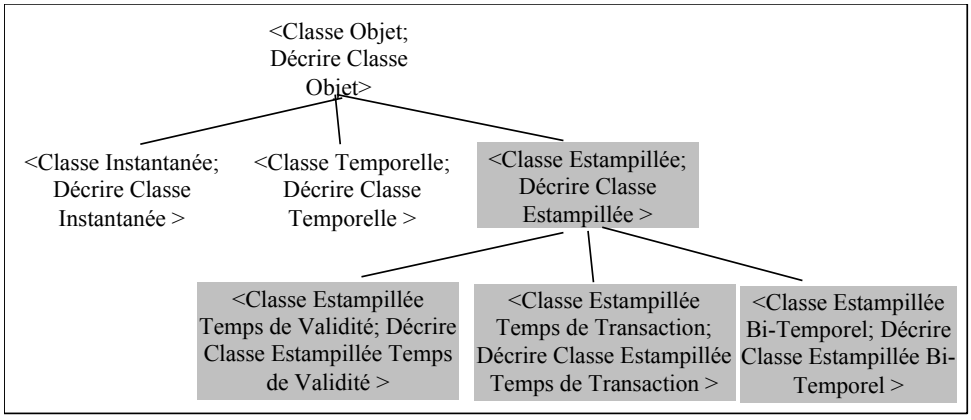


Figure 217: Partie de l'arbre de processus de la méthode O* étendue

21.11 Insertion du concept d'estampillage des stimuli externes dans O*

21.11.1 Problème pris en compte

L'arrivée de stimuli externes reconnus par l'application nécessite le fait de les situer sur l'axe du temps selon leur temps de validité. C'est le cas pour les organisations qui exécutent des activités sensibles en avance ou avec un certain délai. Par exemple, le stimulus *Augmenter le salaire d'un employé* induit différents traitements selon son temps de validité. En effet, si le traitement est fait en avance, le changement de salaire doit être pris en compte seulement lorsque son temps de validité devient la date courante, alors que si le changement est fait en retard, cela demande un paiement correctif.

Les modèles OO devraient donc fournir des concepts spécifiques concernant ces trois différents types de stimulus externe selon leur estampillage dans le temps. En effet, selon l'utilité de montrer que les stimuli ont une nature de temps dans le passé, le présent ou le futur, l'ingénieur peut avoir besoin de les différencier. Un stimulus externe matérialise l'émission d'un message de l'environnement vers

l'application. Cette dernière répond à ce message en exécutant un traitement spécifique. Le temps de validité d'un stimulus externe est le temps de validité associé au message. Trois cas doivent être considérés :

(a) un message reçu en avance : Parfois, l'analyste veut autoriser le déclenchement d'un traitement en avance. Cependant, pour éviter les problèmes relatifs à la gérance des post-actions (pour maintenir la cohérence de la base de données à travers le temps), même si l'acquisition du message est faite en avance, le traitement associé au stimulus est exécuté à l'heure. Un stimulus *A Priori* a un temps de validité dans le futur.

(b) un message reçu en retard : De la même façon, l'analyste peut vouloir corriger la modélisation du monde réel stockée dans la base de données (ceci étant limité à quelques cas exceptionnels). Un stimulus *A Posteriori* a un temps de validité dans le passé. La correction du compte d'un client par rapport à un chèque enregistré dans la banque est un exemple de stimulus *A Posteriori*. Le traitement associé devra corriger la situation en effaçant le traitement fait avec le mauvais compte et appliquer le traitement avec le compte correct.

(c) un message reçu à l'heure : Si le stimulus externe n'est pas un stimulus d'anticipation ni un stimulus en retard, alors, c'est un stimulus qui agit sur la base de données exactement lorsqu'il se produit (ce sont ces stimuli qui sont les plus courants). Un stimulus *In Time* a un temps de validité dans le temps présent.

21.11.2 Choix du patron d'extension

L'insertion de concepts permettant l'estampillage de stimuli externes dans une méthode OO se représente dans la carte d'extension par six sections ayant la même intention source *(Démarrer)*. Le fait d'avoir six sections ayant les mêmes intentions source et cible illustre la diversité des stratégies permettant d'accomplir cette intention cible, comme le visualise la Figure 218.

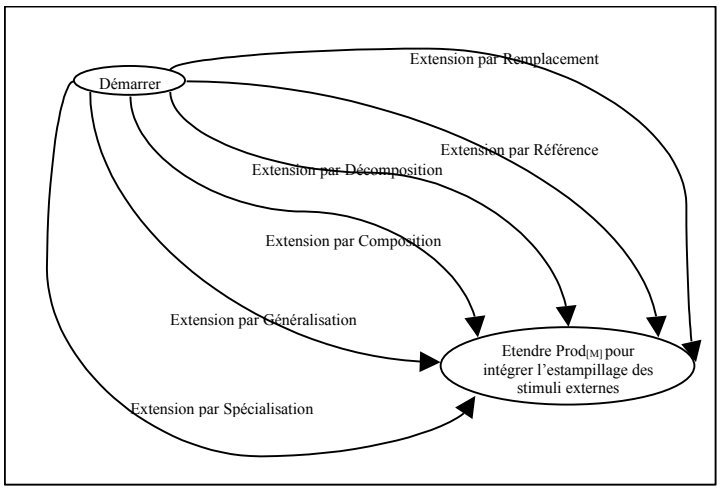


Figure 218: Sections de la carte d'extension d'une méthode aux applications temporelles ayant comme intention cible l'insertion de l'estampillage des stimuli externes

Le choix de la stratégie particulière à appliquer dans le cas de O* dépend de la situation de cette méthode avant extension.

Le modèle de la méthode O* possède déjà un concept permettant de représenter les stimuli externes appelé *Evénement Externe*. La Figure 219 décrit ce concept comme une spécialisation du concept d'*Evénement*, ainsi que ceux d'*Evénement Interne* et d'*Evénement Temporel*. Un *Evénement Externe* est relié à un message envoyé par un acteur spécifique du système. Une instance de ce type d'événement se produit lorsque le système reçoit une occurrence d'un message, reconnaît son type, et que son prédicat est *vrai*. Le prédicat d'un *Evénement Externe* est composé de conditions de vérification sur le contenu de son message. Une fois ces vérifications effectuées, le contenu du message est accepté. On peut donc constater que ce concept ne permet pas l'estampillage des stimuli dans le temps.

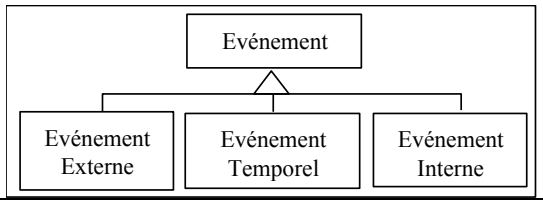
Description du **PRODUIT** avant extension	Représentation du **PRODUIT** avant extension
Evénement = **Spécialisation** (Evénement Externe, Evénement Interne, Evénement Temporel)	

Figure 219 : Partie du PRODUIT de O* avant l'application du patron

Ce concept d'*Evénement Externe* n'est pas défini pour prendre en compte l'estampillage, et est donc un équivalent au concept d'*Evénement Externe In Time*. En conséquence, ce concept peut être généralisé pour permettre la spécialisation aux deux types d'événements estampillés. L'ingénieur de méthodes va alors appliquer sur ce modèle la stratégie d'extension par GENERALISATION pour pouvoir choisir les stimulus externes selon les trois types définis précédemment : *A priori*, *In Time* et *A posteriori*.

L'ingénieur de méthodes exécute donc le patron d'extension décrit dans l'annexe B et dont l'intention est d'*« Etendre, par GENERALISATION, le concept d'Evénement Externe pour intégrer le concept d'estampillage de stimuli externes ».*

21.11.3 Instanciation du patron d'extension à la méthode O*

L'interface du patron d'extension instancié pour la méthode O* est illustrée dans la figure suivante.

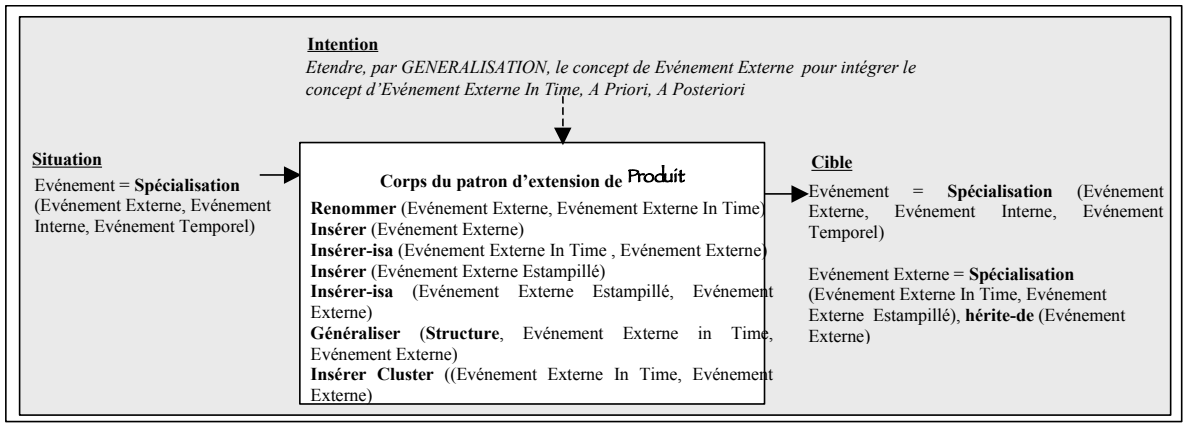


Figure 220: Application du patron d'extension de O* permettant de remplacer le concept d'estampillage des stimuli externes

L'application de ce patron permettra donc la création de différents types d'*Evénement Externe*. Si l'on reprend l'exemple du salaire d'un employé à modifier, les événements suivants illustrent les trois types définis précédemment : *Modification de salaire en avance* est un exemple d'*Evénement Externe A Priori*, *Report de modification du salaire* est un *Evénement Externe A Posteriori* et M*odification du salaire* est un *Evénement Externe In Time*.

Le corps de ce patron est composé de plusieurs opérateurs permettant d'effectuer cette généralisation et de différencier les événements externes n'ayant pas de sémantique temporel (*In Time*) des deux autres types d'événements externes, ainsi que le montre la Figure 221 suivante.

Corps du Patron	Description du **PRODUIT** après extension
Renommer (Evénement Externe, Evénement Externe In Time) **Insérer** (Evénement Externe) **Insérer-isa** (Evénement Externe In Time , Evénement Externe) **Insérer** (Evénement Externe Estampillé) **Insérer-isa** (Evénement Externe Estampillé, Evénement Externe) **Généraliser** (**Structure**, Evénement Externe in Time, Evénement Externe) **Insérer Cluster** ((Evénement Externe In Time, Evénement Externe)	Evénement = **Spécialisation** (Evénement Externe, Evénement Interne, Evénement Temporel) Evénement Externe = **Spécialisation** (Evénement Externe In Time, Evénement Externe Estampillé), **hérite-de** (Evénement Externe)

Figure 221: Partie du PRODUIT de O* après l'application du patron

On peut voir sur la Figure 222 que la partie **PRODUIT** de la méthode O* a été étendue pour intégrer une nouvelle définition du concept d'*Evénement Externe* pour permettre à l'ingénieur de méthodes de définir plusieurs spécialisations possibles de ce concept selon l'estampillage qui lui est rattaché.

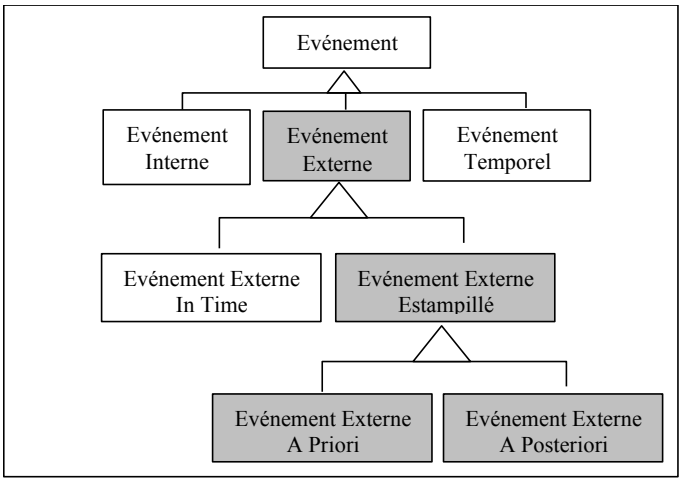


Figure 222: Partie du méta-modèle de la méthode O* étendue

L'insertion de ce nouveau concept d'*Evénement Externe* intègre également sa spécialisation d'*Evénement Externe Estampillé* qui est lui-même la généralisation des concepts d'*Evénement Externe A Priori* et d'*Evénement Externe A Posteriori*.

21.12 Insertion de la construction de l'estampillage des stimuli externes dans O*

21.12.1 Problème pris en compte

L'ingénieur de méthodes a intégré le concept d'estampillage des stimuli externes dans la partie **PRODUIT** de la méthode d'origine et il souhaite maintenant étendre la partie **DÉMARCHE** de celle-ci.

21.12.2 Choix du patron d'extension

Comme le visualise la Figure 223, l'insertion de la construction des concepts permettant l'estampillage des stimuli externes est représentée dans la carte d'extension comme l'intention cible de trois sections différentes représentant chacune un patron d'extension spécifique. Il y a donc trois stratégies différentes possibles parmi lesquelles l'ingénieur de méthodes doit choisir.

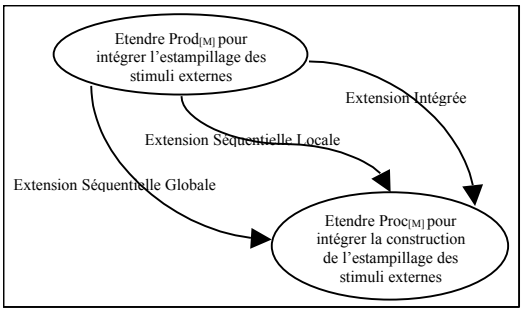


Figure 223: Sections de la carte d'extension d'une méthode aux applications temporelles ayant comme intention cible l'insertion de la construction de l'estampillage des stimuli externes

Le choix de la stratégie à exécuter, et par là même de la section à réaliser (patron à instancier), est dépendant de la situation de O*.

Le modèle de cette méthode possède maintenant le concept d'*Evénement Externe* qui a été spécialisé en trois autres concepts appelés *Evénement Externe In Time* et *Evénement Externe Estampillé* par la stratégie GENERALISATION. Cependant, la partie **DÉMARCHE** de O* n'a pas encore pris cette modification en compte. La Figure 224 décrit, de manière textuelle et graphique, la partie de la **DÉMARCHE** de O* avant extension.

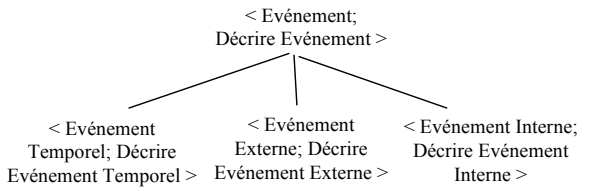
Description de la **DÉMARCHE** avant extension	Représentation de la **DÉMARCHE** avant extension
<Evénement; Décrire Evénement> = " <Evénement Interne; Décrire Evénement Interne> " <Evénement Temporel; Décrire Evénement Temporel> " <Evénement Externe ; Décrire Evénement Externe>	

Figure 224 : Partie de la DÉMARCHE de O* avant l'application du patron

Différentes stratégies d'extension de la **DÉMARCHE** sont possibles après cette stratégie d'extension du **PRODUIT.** Comme l'ingénieur de méthodes a commencé son exécution de l'extension de la **DÉMARCHE** de O* par la stratégie INTEGREE, il paraît assez logique de continuer ainsi et d'intégrer toute modification de la **DÉMARCHE** de cette manière. L'ingénieur de méthodes doit ensuite décider s'il préfère intégrer cette construction comme une spécialisation par type ou par état. Pour des raisons de simplicité lors de la modification de la partie **DÉMARCHE**, nous choisirons la spécialisation par type et la stratégie à appliquer pour cette extension spécifique sera donc celle de l'extension par GENERALISATION INTEGREE (par type).

L'ingénieur de méthodes exécute donc le patron d'extension décrit dans l'annexe B et dont l'intention est *« Etendre, par GENERALISATION INTEGREE (par type), la construction du concept d'Evénement Externe pour intégrer la construction de l'estampillage des stimuli externes ».*

21.12.3 Instanciation du patron d'extension à la méthode O*

L'interface de ce patron d'extension instancié pour la méthode O* est la suivante.

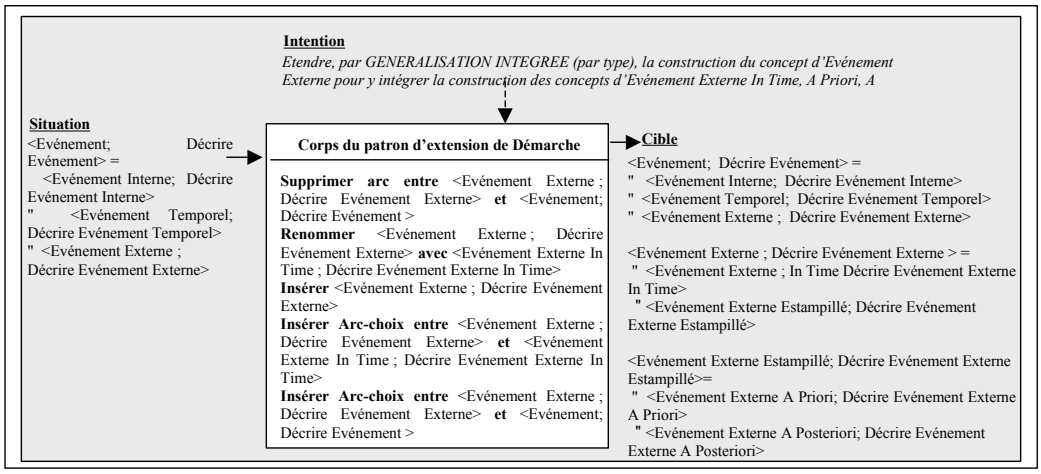


Figure 225: Application du patron d'extension de la méthode O* permettant d'intégrer la construction de l'estampillage des stimuli externes.

Le résultat de l'application de ce patron sur la méthode O* permet donc d'intégrer la démarche de description des événements externes estampillés dans la **DÉMARCHE** de O*. La branche concernée est insérée entre la démarche de description de l'événement externe et son père dans l'arbre de processus, ce qui permet de réaliser les démarches de spécialisation.

Le corps de ce patron contient plusieurs opérateurs permettant cette modification de la partie **DÉMARCHE** de O*. Ces opérateurs sont décrits dans la Figure 225. Dans la même figure est présentée la description de la **DÉMARCHE** après exécution de ces opérateurs.

Corps du Patron	Description de la **DÉMARCHE** après extension
Renommer <Evénement Externe ; Décrire Evénement Externe> **avec** <Evénement Externe In Time ; Décrire Evénement Externe In Time> **Insérer** <Evénement Externe ; Décrire Evénement Externe> **Insérer Arc-choix entre** <Evénement Externe ; Décrire Evénement Externe> et <Evénement Externe In Time ; Décrire Evénement Externe In Time>	<Evénement; Décrire Evénement> = <Evénement Interne; Décrire Evénement Interne> " <Evénement Temporel; Décrire Evénement Temporel> " <Evénement Externe ; Décrire Evénement Externe> <Evénement Externe ; Décrire Evénement Externe > = <Evénement Externe ; In Time Décrire Evénement Externe In Time> " <Evénement Externe Estampillé; Décrire Evénement Externe Estampillé> <Evénement Externe Estampillé; Décrire Evénement Externe Estampillé>= <Evénement Externe A Priori; Décrire Evénement Externe A Priori> " <Evénement Externe A Posteriori; Décrire Evénement Externe A Posteriori>

Figure 226: Partie de la DÉMARCHE de O* après l'application du patron

On peut donc constater à la figure suivante que la partie **DÉMARCHE** de la méthode O* a été étendue de manière à prendre en compte la construction de ces nouveaux concepts représentant les spécialisations du concept d'*Evénement Externe*.

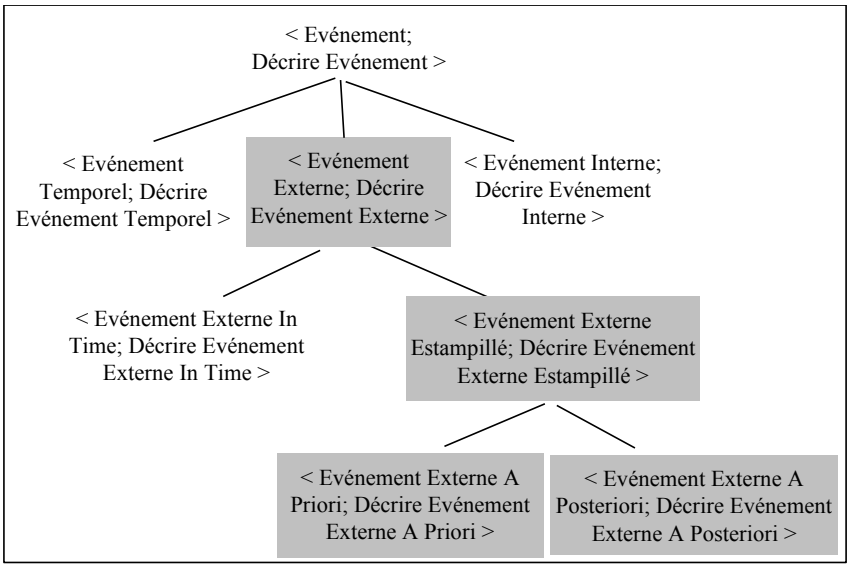


Figure 227: Partie de l'arbre de processus de la méthode O* étendue

La greffe de la branche de la démarche de description du nouvel événement externe permet d'insérer également dans la **DÉMARCHE** de O* toutes les spécialisations qui en découlent, c.-à-d. la démarche des événements estampillés et par là même celles des événements externe estampillés en avance ou en retard.

21.13Extension de la méthode O*

21.13.1 Méthode O* d'origine

La méthode O* d'origine contient plusieurs parties qui vont être modifiées par l'exécution de la carte.

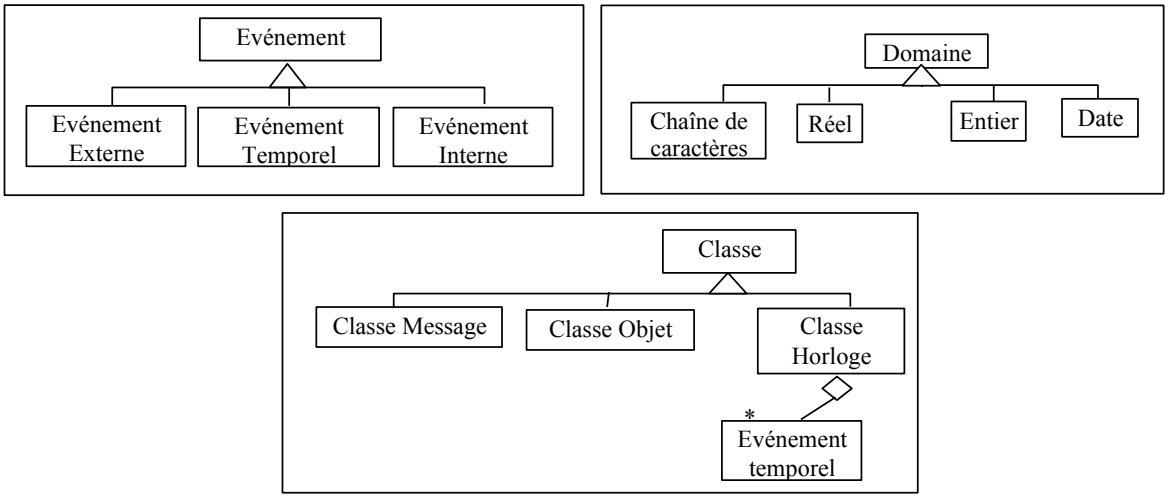


Figure 228: Parties du PRODUIT de la méthode O* avant extension

La partie **PRODUIT** de O* possède :

- le concept d'Evénement spécialisable en trois types : Evénement Externe, Evénement Interne et Evénement Temporel.
- le concept de Domaine spécialisable en quatre types : *Chaîne de caractères*, *Réel*, *Entier* et *Date*.
- le concept de *Classe* spécialisable en trois types de classe différents : *Classe Message*, *Classe Objet* et *Classe Horloge*, elle-même composée d'un ensemble d'*Evénement Temporel*.

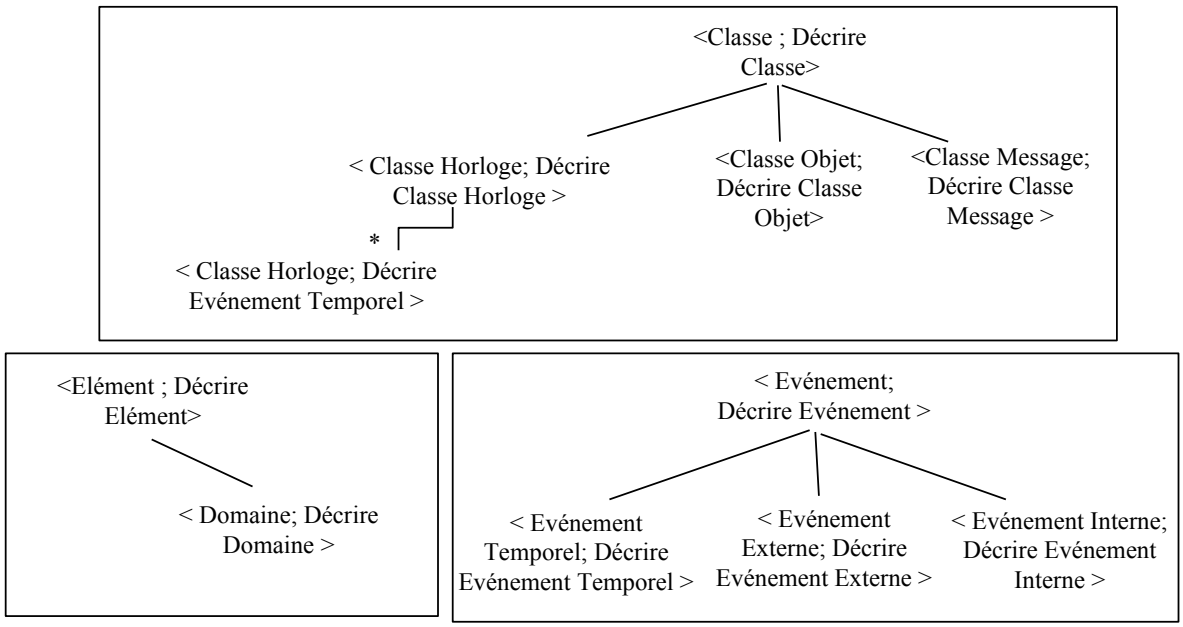


Figure 229: parties de la DÉMARCHE de la méthode O* avant extension

La partie **DÉMARCHE** de la méthode O* prend en compte les démarches de description :

- Du concept de *Classe* qui est un contexte choix possédant trois possibilités, c.-à-d. les démarches de description de *Classe Horloge*, *Classe Objet* et *Classe Message*.
- Du concept de *Domaine*.
- Du concept d'*Evénement* étant un contexte plan possédant trois possibilités, c.-à-d. les démarches de description de *Evénement Temporel*, *Evénement Externe* et *Evénement Interne*.

21.13.2 Chemin de navigation choisi pour étendre la méthode O* aux applications temporelles

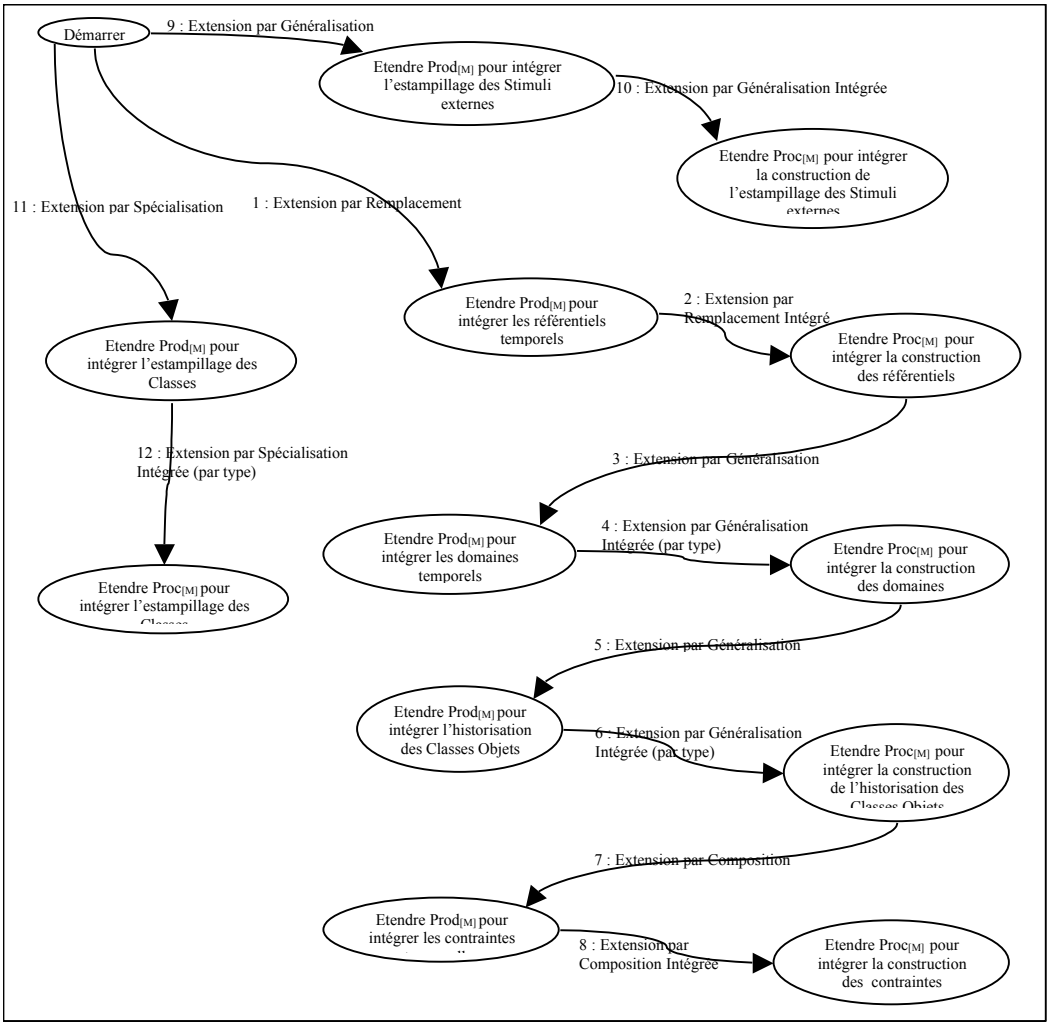


Figure 230: Chemin suivi pour O* dans la carte d'extension d'une méthode aux applications temporelles